LatticeMico Dual Boot

The LatticeMico Dual Boot is used to reconfigure the Analog Sense and Controls (ASCs) with golden (fail-safe) images obtained from an external SPI flash. This reconfiguration is performed only once at power-up. Once the reconfiguration is complete, the component returns to a dormant state, at which time all other components are free to communicate with the ASCs via the I^2C interface.

Version

This document describes the 1.0 version of the LatticeMico Dual Boot.

Features

The LatticeMico Dual Boot IP must be used together with the Embedded Functional Block (EFB) of the MachXO2 or Platform Manager 2 device, and has the following features to support the dual boot task. The component assumes that at least one ASC image is available at a user-configurable location in the SPI flash.

- Reconfigures the ASCs with golden (fail-safe) images obtained from flash memory.
- Supports reconfiguration of up to eight ASCs.
- Supports both external SPI Flash and User Flash Memory via EFB.
- Skips over undiscovered ASCs, if required. If an ACK is not returned by any ASC on the I^2C bus, the ASC is said to be undiscovered. The Dual-Boot component will check for an ASC only once.

Copyright © August 2013 Lattice Semiconductor Corporation.

- Supports termination without releasing busy signal, so that other components detect that ASCs have not been reconfigure correctly.
- Configurable to either hardware-only components or software assisted components.
- WISHBONE interface to EFB and LatticeMico8 microcontroller.
- Supports synchronization with LatticeMico Mutex to prevent background programming while reconfiguration is in progress.

Functional Description

The Dual Boot IP can be used as a WISHBONE Master or a WISHBONE Slave. When in the WISHBONE Master mode, the IP is self-contained and does not require a microcontroller to function. When in the WISHBONE Slave mode, the IP requires the LatticeMico8 microcontroller to be used together to complete the function. Figure 1 and Figure 2 give an overview of the IP applications based on the mode it is selected.

Figure 1: Dual Boot WISHBONE Master Mode Block Diagram

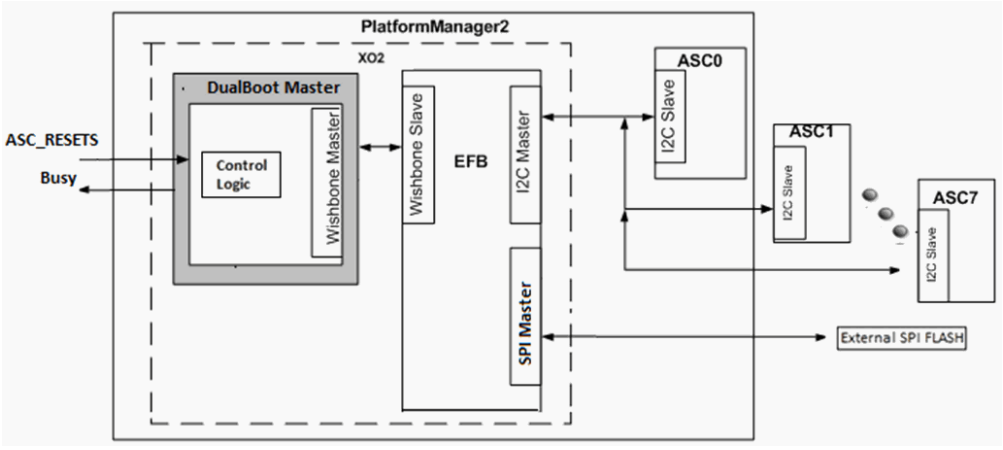
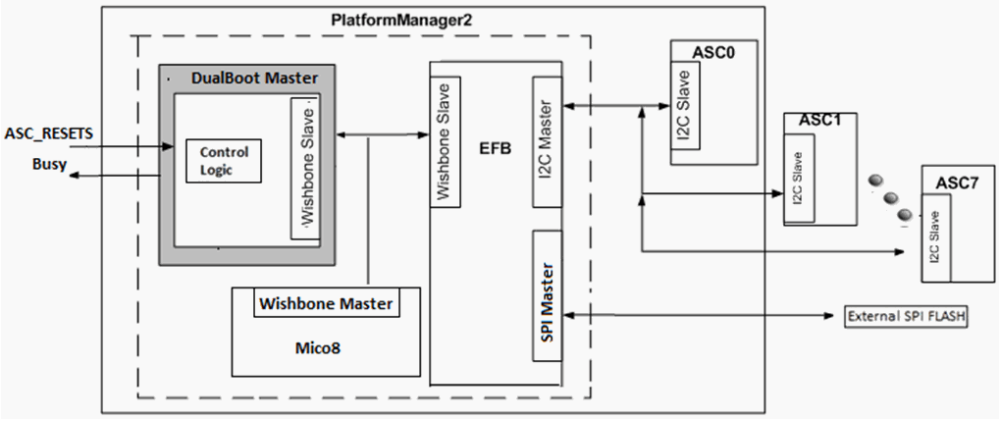


Figure 2: Dual Boot WISHBONE Slave Mode Block Diagram



WISHBONE Master Mode

When in WISHBONE Master mode, the Dual-Boot algorithm is implemented using RTL. It is independent and implements the algorithm in hardware for reading the ASC image(s) off the external SPI flash and programming to the shadow register file of the respective ASCs. The Dual-Boot algorithm contains:

1. WISHBONE master interface communicates with the WISHBONE slave interface of the EFB (in order to drive data on the primary I^2C and SPI).
2. SPI flash sequencer reads the hexadecimal images of the ASC off the external SPI flash via the EFB SPI interface and interprets it.
3. I^2C sequencer programs the ASCs via the EFB I^2C interface. I^2C should be configured to operate at 400KHz.

WISHBONE Slave Mode

When in WISHBONE Slave mode, the Dual-Boot algorithm is implemented using the LatticeMico8 microcontroller. This algorithm for reading the ASC image(s) from the external SPI flash. Programming to the shadow register file of the respective ASCs is performed by the LatticeMico8 microcontroller. The Dual-Boot component contains a WISHBONE slave interface and a memory-mapped register file that allows the LatticeMico8 microcontroller to query the SPI flash opcode table and EFB chip select ID.

Note

The Dual-Boot algorithm will be invoked only once during a power cycle. A Watchdog timer expiration will not result in the reconfiguration of the ASCs.

Note

Watchdog is not used during Dual-Boot operation.

Programmer/Bit-Generation Requirements

The Dual-Boot component places certain requirements on Diamond Programmer and other third-party programmer. The following sections describe these requirements.

Layout of ASC Images in SPI Flash

A uniform layout is required to ensure that the size of the Dual-Boot IP is minimal and LUTs are not used to hunt for the ASC images. Table 1 describes the key features required within the layout[1] of ASC images that reside within the SPI flash. Each ASC is assigned a 256 byte region in SPI flash. All ASC regions must be placed in consecutive 128 byte aligned addresses.

Table 1: ASC Image Format

Element Number	Name	Value	Size	Description
0	HEADER	0xC5	8-bit	A header to identify if an ASC image exists or not.
1	ASC TYPE	0x00	8-bit	One byte to identify whether the type of ASC.
2	I^2C SLAVE ADDRESS	Slave Address[7..0]	8-bit	One byte to identify the ASC's 7-bit I^2C slave address. [7..0] = [i2cdonebit, i2cslaveaddr[6:3], xxx]
3	ASC DATA	Data	896-bit	Image to be loaded into ASC Shadow Register File.
4	CRC	CRC[15..0]	16-bit	16-bit CRC. It is preferable to use existing 16-bit frame data CRC algorithm used in other Lattice devices since the software infrastructure already exists.
5	USERCODE	USERCODE[31..0]	32-bit	32-bit USERCODE.
6	PADDING	1	1080-bit	Page bound padding bits.

Erase/Program Protocol

The Dual-Boot component is normally only part of the golden (fail-safe) design that is located in SPI flash. This design can be used to bring up a system that has never been programmed, or a system in which the MachXO2/ Platform Manager 2 or ASCs were programmed with errors. While it is possible that the programming of the on-chip configuration flash of MachXO2/ Platform Manager 2 or EEPROM or ASC may have resulted in programming errors, it is important to note that the golden design is automatically loaded only in case of programming errors in the MachXO2/Platform Manager 2 on-chip configuration flash. Thus, it is the responsibility of the programmer to ensure that the following algorithm is used while erasing and programming the MachXO2/Platform Manager 2 and ASCs within the system:

1.The format of the actual layout is determined during bit-generation. All of the features mentioned in this section should be present so that the Dual-Boot component can perform its intended function.

Step 1: Erase PROGRAMN fuse in MachXO2/Platform Manager 2.

Step 2: Erase, program, and verify EEPROM of ASCs.

Step 3: Erase, program, and verify Configuration Flash of MachXO2/Platform Manager 2.

Step 4: Set PROGRAMN fuse in MachXO2/Platform Manager 2.

Runtime Requirements

The ASC can accept I^2C commands once it is configured. The ASC will indicate that it is ready to receive I^2C commands by asserting the ASC_RESETS signal. This signal is transmitted from the ASC to the ASCVM soft IP on MachXO2/Platform Manager 2 via the 3Wire interface and made available to the local signal pool. User is responsible for wiring each ASC's ASC_RESETS signal to the Dual-Boot IP when it instantiates the WISHBONE sub-system that contains the Dual-Boot IP.[1]

The Dual-Boot component will wait until the ASC_RESETS of each mandatory ASC is asserted before commencing reconfiguration of all the ASCs. The Dual-Boot component will not wait for the ASC_RESETS of non-mandatory (i.e., hot-swappable) ASCs prior to commencing reconfiguration. It is the responsibility of the ASCVM IP to ensure that the ASC_RESETS associated with each ASC is de-asserted at power-up (i.e. default at reset). When the golden (fail-safe) design is reconfiguring the ASCs, the ASCs should not be accessed by any other soft IP in the XO2 via the I^2C or 3Wire interface. The Dual-Boot component contains an output port whose purpose is to let other IPs know when it is safe to access the ASCs. The Dual-Boot component asserts (i.e., logic 1) this output port when it is busy programming the ASCs. It de-asserts (i.e., logic 0) this output port when programming is complete indicating that other components can now access the ASCs.

Note

The following soft IPs should monitor the aforementioned output port: ASCVM and VID (only in WBM configuration). User should wire this output port of the Dual-Boot to the ASCVM, VID, Fault Logger, and other components.

Note

In one power cycle the Dual-Boot component will reconfigure the ASCs only once. Any subsequent de-assertion of ASC_RESETS will not result in Dual-Boot reconfiguring the ASCs all over again. This also means that the Dual-Boot component stops monitoring ASC_RESETS once it has completed the first (and only) reconfiguration of ASCs.

1.The AGOOD input ports of the Dual-Boot IP will also appear as input ports in the WISHBONE sub-system's top level RTL module.

Dual-Boot Operation

The operation of the Dual-Boot component is identical in both hardware and software configurations. It can be in one of two states: active or dormant. The component enters active state on reset (or power-up). Once it has finished reconfiguring the ASC(s), the component enters the dormant state. It will not leave this state until the next reset (or power-up). This ensures that the reconfiguration of the ASCs is performed only once during any power or reset cycle. The following steps explain the operation of the component within the active state.

Step 1: Signal to all IPs/components that ASCs are being reconfigured. Wait for ASC_RESETS of all mandatory ASCs to be asserted, guaranteeing that I^2C commands can be actually accepted by the ASCs.

Step 2: Initialize SPI flash address pointer to base of first ASC image. If header is not found, go to Step 14.

Step 3: Request access to XO2's Primary I^2C from the I^2C Mutex component. Repeat until access is granted.

Step 4: Set ASC's I^2C slave address from SPI flash.

Step 5: Set ASC type from SPI flash. Query ASC's MFG ID using 'ReadID' command. If ASC not found at given I^2C slave address or if ASC type does not match, go to Step 6. Else go to Step 7.

Step 6: Set current ASC's skip-on-fail bit from SPI flash. If this bit is set, go to Step 11. Else go to Step 14.

Step 7: Check if ASC is busy by querying ASC's Status Register using 'ReadStat' command. The Dual Boot algorithm performs this step to ensure that the ASC is not busy with other operations that will disallow it from accepting a reconfiguration command. Repeat until ASC is not busy.

Step 8: Read ASC image (including CRC) from SPI flash and program to Shadow Register file of ASC using 'WrAllReCfg' command. The total number of bytes programmed to ASC depends on the type: 93 bytes (includes 2 byte CRC).

Step 9: Read status register in ASC to determine if a CRC error was detected. A CRC error indicates programming error and Step 8 is repeated.

Step 10: Make the new contents of Shadow Register file current using 'LdReCfg' command.

Step 11: Set current ASC's final bit from SPI flash. This bit indicates if any additional ASCs need to be reconfigured. Go to Step 13 if this bit is set. Else, go to Step 12.

Step 12: Reposition SPI flash address pointer to base of next ASC image and go to Step 3.

Step 13: Signal to all components that they can begin communication with ASCs.

Step 14: Enter dormant state.

Note

If ASC_RESETS of any ASC is de-asserted during any steps 2 through 12, the Dual-Boot component will skip Step 13 and transition to Step 14.

Configuration

The following sections describe the graphical user interface (UI) parameters, the hardware description language (HDL) parameters, and the I/O ports that user can use to configure and operate the LatticeMico Dual Boot.

UI Parameters

Table 2 shows the UI parameters available for configuring the LatticeMico Dual Boot through the Mico System Builder (MSB) interface. For more information refer to the Platform Designer documentation in Diamond online help.

Table 2: Dual Boot UI Parameters

Dialog Box Options	Description	Allowable Values	Default Value
Master Mode	Specifies the Dual Boot as master mode.	Selected | Not selected	Selected
Slave Mode	Specifies the Dual Boot as slave mode	selected | not selected	not selected
Instance Name	Specifies the name of the Dual Boot instance.	Alphanumeric and underscores	dualboot
Base Address	Specifies the base address for configuring the Dual Boot, which requires for Dual Boot Slave Mode. The minimum boundary alignment is 0x80.	0X80000000–0XFFFFFFFF	0X80000000
EFB Setting			
EFB Base Address	Specifies the base address of the EFB component, which requires for Dual Boot Master Mode. The minimum boundary alignment is 0x80.	0X80000000–0XFFFFFFFF	0X80000000
SPI Chip Select Line	Specifies the number of ASC Log	1 – 8	1

Table 2: Dual Boot UI Parameters (Continued)

Dialog Box Options	Description	Allowable Values	Default Value
Mutex Setting			
Mutex Slave Address	Specifies the base address of the Mutex component, which requires for the Dual Boot Master Mode. The minimum boundary alignment is 0x80.	0X80000000–0XFFFFFFFF	0X80000000
SPI Flash Setting			
Flash Base Address	Specifies the base address of the SPI Flash, which requires for the Dual Boot Master Mode.	0x00000000–0XFFFFFFFF	0x00000000
Read	Specifies the opcode for external SPI Flash Read (READ)	0 - 255	3
ASC Setting			
Number of ASCs	Specifies the number of the ASCs in the design	1 - 8	1
Optional ASCs	A bit-mask that specifies the optional ASCs that skip the dualboot operation. Each bit corresponding to a ASC, up to 8 ASCs.	0x00	0xFF

HDL Parameters

Table 3 lists the parameters that appear in the HDL.

Table 3: Dual Boot HDL Parameter

Parameter Name	Description	Allowable Values
DB_MASTER	Indicates how the Dual-Boot component is being configured. 1 means it is begin configured as DB-H and 0 means it is being configured as DB-S.	0, 1
DB_EFB_BASE_ADDRESS	Indicates the 32-bit WISHBONE address at which the MachXO2/Platform Manager 2 EFB resides.	0 to (2^{32} – 1)
DB_CHIP_SELECT	Indicates which chip select within the EFB SPI corresponds to the Dual-Boot SPI flash.	1, 2, 4, 8, 16, 32, 64, or 128
DB_SPI_READ_OPCODE	The SPI flash read opcode (from SPI flash datasheet)	0 to 255
DB_SPI_OFFSET	The offset within the SPI flash at which the first ASC image can be found.	0 to (2^{32} – 1)
DB_ASC_COUNT	The total number of ASCs in the design.	1 to 8

I/O Ports

Table 4 describes the input and output ports of the LatticeMico Dual Boot.

Table 4: Dual Boot I/O Ports

I/O Port	Direction	Active	Description
System Clock and Reset			
CLK	I	—	System Clock
RST	I	Low	System Reset
WISHBONE Master Signal			
DBM_DAT_I	I	—	8-bit data used to read a byte of data from a specific register in the register
DBM_ACK_I	I	High	Transfer acknowledge signal asserted, indicates the requested transfer is acknowledged.
DBM_ERR_I	I	—	Not used, always tied to 0
DBM_RTY_I	I	—	Not used, always tied to 0
DBM_CYC_O	O	High	Indicates a valid bus cycle is present on the WISHBONE bus.
DBM_STB_O	O	High	Indicates the WISHBONE slave is the target for the current transaction on the bus.
DBM_WE_O	O	—	Level sensitive Write/Read control signal. Low - Read operation, High - Write operation
DBM_ADR_O	O	—	32-bit wide address used to select a specific register
DBM_DAT_O	O	—	8-bit data used to read a byte of data from a specific register
DBM_CTI_O	O	—	Not used, always tied to 0
DBM_BTE_O	O	—	Not used, always tied to 0
DBM_LOCK_O	O	—	Not used, always tied to 0
DBM_SEL_O	O	—	Not used, always tied to 0
WISHBONE Slave Signal			
DBS_CYC_I	I	High	Indicates a valid bus cycle is present on the bus.
DBS_STB_I	I	High	Asserts an acknowledgment in response to the assertion of the WISHBONE Master strobe.
DBS_WE_I	I	—	Level sensitive Write/Read control signal. Low - Read operation, High - Write operation
DBS_ADR_I	I	—	32-bit wide address used to select a specific register

Table 4: Dual Boot I/O Ports (Continued)

I/O Port	Direction	Active	Description
DBS_DAT_I	I	—	8-bit data used to read a byte of data from a specific register
DBS_CTI_I	I	—	Not used, always tied to 0
DBS_BTE_I	I	—	Not used, always tied to 0
DBS_LOCK_I	I	—	Not used, always tied to 0
DBS_SEL_I	I	—	Not used, always tied to 0
DBS_DAT_O	O	—	8-bit data used to read a byte of data from a specific register
DBS_ACK_O	O	High	Indicates the requested transfer is acknowledged.
DBS_ERR_O	O	—	Indicates the address is incorrect
DBS_RTY_O	O	—	Not used, always tied to 0
Other signals			
ASC_RESETS	I	High	Indicated the ASC is ready to reset when asserted. Each bit correspond to a ASC device
asc_write_enable	O	—	Indicated the ASC is write enabled
db_busy	O	High	Indicates that the Dual Boot is busy and in the process of reconfiguring the ASCs.

Register Descriptions

The LatticeMico Dual Boot WISHBONE Slave module has a register map to allow the service of the hardened functions through the WISHBONE bus interface read/write operations. Table 5 and Table 6 describe the register map of the Dual Boot WBS module.

Table 5: WISHBONE Addressable Registers for Dual Boot Module

Register Name	Register Function	Address	Access
ASCCOUNT	Holds the info of the number of ASCs	0x0	Read
ASCOPT	Holds the info of the optional ASCs	0x1	Read
FLASHBASE_LO	Holds the external SPI Flash Address[7:0]	0x2	Read
FLASHBASE_HI	Holds the external SPI Flash Address[15:8]	0x3	Read
SPI_RD_OPCODE	Holds the opcode for SPI Flash Read (READ)	0x4	Read
SPI_CHIP_SEL	Holds the chip select line of the SPI Flash	0x5	Read
AGOOD_STS	Status of the ASCs device	0x6	Read
CONTROL	Dual Boot Control Register	0x7	Read/Write

ASC Count Register Definition - ASCCOUNT

ASCCOUNT is an 8-bit register, which holds the info of the number of ASCs in the design. This value that can have values 1 through 8.This register correlates to Verilog parameter ASC_COUNT. The WISHBONE host has Read-Only access to these registers.

Optional ASC Register Definition - ASCOPT

ASCOPT is a 8-bit register, which holds the info of the optional ASCs in the design.

This register correlates to Verilog parameter ASC_OPT. The WISHBONE host has Read-Only access to these registers.

SPI Address Register Definition - FLASHBASE_LO/ FLASHBASE_HIGH

FLASHBASE _LO / FLASHBASE _HI are 8-bit registers, which combined to hold the external SPI Flash Address. This register correlates to Verilog parameter FLASH_BASE_ADDRESS. The WISHBONE host has Read access to this register.

FLASHBASE_LO register holds the SPI Address Register Status of the lower 8-bit value [7:0].

FLASHBASE_HI register holds the SPI Address Register Status of the upper 8-bit value [15:8].

SPI Flash Read Command Register Definition - SPI_RD_OPCODE

SPI_RD_OPCODE are 8-bit registers, which hold a specific SPI Flash command and. correlates to a corresponding Verilog parameter. The WISHBONE host has Read-Only access to these registers.

SPI Flash Chip Select Register Definition - SPI_CHP_SEL

SPI_CHP_ SEL is an 8-bit register that specifics which chip select line (LatticeMico EFB SPI Master) is connected to the external SPI flash. This register correlates to Verilog parameter EFB_CHIP_SELECT.The WISHBONE host has Read-Only access to these registers.

ASC Status Register Definition - AGOOD_STS

AGOOD_STS is a 8-bit register that indicates whether an ASCs ASC_RESET is asserted or not. The maximum number of ASCs is eight and one bit is used per ASC. The WISHBONE host has Read-Only access to these registers.

Control Register Definition - CONTROL

The WISHBONE host has Read and Write access to these registers except bit 4 - DB_BUSY.

Table 6: Control Register Bit Definition

Bit	Field	Description	
0	BUSY	To indicate if Dual Boot is in progress 1 – Busy, 0 – Not busy	Read/Write
1	ASC_WRITE_ENABLE	To enable or disable the ASC Write 1 – Enable, 0 - Disable	Read/Write
7:2	RSVD	Reserved Bit	N/A

LatticeMico8 Microcontroller Software Support

This section describes the LatticeMico8 microcontroller software support provided for the LatticeMico Dual Boot component.

Device Driver

The Dual Boot device driver interacts directly with the Dual Boot instance. This section describes the limitations, type definitions, structure, and functions of the Dual Boot device driver.

Type Definitions

This section describes the type definitions for the Dual Boot device context structure. This structure, shown in Figure 3, contains the Dual Boot component instance-specific information and is dynamically generated in the DDStructs.h header file. This information is largely filled in by the managed build process by extracting the Dual Boot component-specific information from the platform specification file. As part of the managed build process, designers can choose to control the size of the generated structure, and hence the software executable, by selectively enabling some of the elements in this structure via C preprocessor macro definitions. These C preprocessor macro definitions are explained later in this document. You should not manipulate the members directly, because this structure is for exclusive use by the device driver. Table 7 describes the parameters of the Dual Boot device context structure shown in Figure 3.

Device Context Structure

Figure 3 shows the Dual Boot device context structure.

Figure 3: Dual Boot Device Context Structure

```
struct st_MicoDBCtx_t {
    const char *   name;
    size_t   base;
    void *   p_mutex;
    unsigned char   i2c_mutex;
} MicoDBCtx_t;
```

Table 7 describes the Dual Boot device context parameters.

Table 7: Dual Boot Device Context Parameters

Parameter	Data Type	Description
name	const char*	Dual Boot instance name (entered in MSB).
base	size_t	MSB-assigned base address for this instance.
p_mutex	void *	This value points to the Mutex instance used by Dual Boot.
i2c_mutex	unsigned char	This value specific the Mutex owner ID for I^2C communication protocol.

C Preprocessor Macro Definitions

This section describes the C preprocessor macro definitions that are available to the software developer. There are two types of macro definitions: 'object-like' and 'function-like'.

The 'object-like' macro definitions do not take any arguments and are used to control the size of the generated application executable. There are three ways an 'object-like' macro definition can be used by the software developer.

1. Manually adding the -D<macro name> option to the compiler's command line in the application's 'Build Properties'. Refer to the *LatticeMico8 Developer User Guide* for more information on how to manually add the macro definition in the application's 'Build Properties' GUI.
2. Automatically adding the -D<macro name> option to the compiler's command-line in the application's 'Build Properties' by enabling the 'check-box' associated with the macro definition. Refer to the *LatticeMico8 Developer User Guide* for more information on how to set up the check/uncheck the macro definitions in the application's 'Build Properties' GUI.
3. Manually adding the macro definition to the C code using the following syntax:

   ```
   #define <macro name>
   ```

It is recommended that the developer use option 1 or 2.

- __MICODB_ENABLE_MUTEX__

  This preprocessor macro definition enables code and data structures for LatticeMico8 Mutex within LatticeMico Dual Boot device driver and application. It is not defined by default.

Table 8: C Preprocessor Function-like Macros For Dual Boot

Macro Name	Second Argument to Macro / Third Argument to Macro (if exist.	Description
MICO_DB_RD_ASCCOUNT	The 8-bit value read from the ASC Count Register	This macro reads a character from the ASC Count Register
MICO_DB_RD_ASCOPT	The 8-bit value read from the ASC optional Register	This macro reads a character from the ASC Optional Register
MICO_DB_RD_FLASHBASE_LO	The 8-bit value read from the lower byte of SPI Address register.	This macro reads a character from the upper byte of SPI Address register
MICO_DB_RD_FLASHBASE_HI	The 8-bit value read from the upper byte of SPI Address register.	This macro reads a character from the upper byte of SPI Address register
MICO_DB_RD_SPI_RD_OPCODE	The 8-bit value read from the SPI Read Command Register	This macro reads a character from the SPI Read Command Register
MICO_DB_RD_SPI_CHIP_SEL	The 8-bit value reads from the SPI Flash Chip Select register.	This macro reads a character to the SPI Flash Chip Select register

Table 8: C Preprocessor Function-like Macros For Dual Boot (Continued)

Macro Name	Second Argument to Macro / Third Argument to Macro (if exist.	Description
MICO_DB_RD_AGOOD_STS	The 8-bit value read from the ASC Status Register	This macro reads a character from the ASC Status Register
MICO_DB_RD_CONTROL	The 8-bit value reads from the control register.	This macro reads a character from the I^2C status register
MICO_DB_WR_CONTROL	The 8-bit value writes to the control register.	This macro writes a character to the I^2C status register

Note: The first argument to the macro is the Dual Boot Slave address.

Functions

This section describes the implemented device-driver-specific functions.

MicoDBInit Function

```
void MicoDBInit (MicoDBCtx_t *ctx);
```

This is the Dual Boot initialization function. Table 9 describes the parameter in the MicoDBInit function syntax.

Table 9: MicoDBInit Function Parameter

Parameter	Description
MicoDBCtx_t	Pointer to a valid MicoDBCtx _t structure representing a valid Dual Boot instance.

MicoDBRegisterMutex Function

```
void MicoDBRegisterMutex (MicoDBCtx_t *ctx,
                          MicoMutexCtx_t *p_mutex,
                          unsigned char i2c_mutex_id);
```

This function registers a Mutex instance into the Dual Boot instance. This Mutex will be used for controlling the EFB I^2C resources to avoid collision.

Table 10 describes the parameter in the MicoDBRegisterMutex function syntax.

Table 10: MicoDBRegisterMutex Function Parameter

Parameter	Description
MicoDBCtx_t	Pointer to a valid MicoDBCtx_t structure representing a valid Dual Boot instance.

Table 10: MicoDBRegisterMutex Function Parameter (Continued)

Parameter	Description
MicoMutexCtx	Pointer to a valid MicoMutexCtx_t structure representing a valid Mutex instance.
unsigned char	Mutex owner ID for I^2C communication protocol.

MicoDB_Reconfigure Function

```
char MicoDB_Reconfigure (MicoDBCtx_t *ctx,
                         MicoEFBCtx_t *p_efb);
```

This function process the Dual Boot Reconfigure Request.

Table 11 describes the parameter in the MicoDB_Reconfigure function syntax.

Table 11: MicoDB_Reconfigure Function Parameter

Parameter	Description
MicoDBCtx_t	Pointer to a valid MicoDBCtx_t structure representing a valid Dual Boot instance.
MicoEFBCtx_t	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

Software Usage Example

This section provides an example of using the Dual Boot. The example is shown in Figure 4 and assumes the presence of a Dual Boot component named “dualboot”, and a EFB component named “efb”.

Figure 4: Dual Boot Software Example

```
#include "MicoUtils.h"
#include "DDStructs.h"
#include "MicoEFB.h"
#include "MicoDualBoot.h"

int main(void){
  MicoDBCtx_t *db = &dualboot_ dualboot;
  MicoEFBCtx_t *efb = &efb_efb;

  size_t db_address = (size_t)(db->base);
  // Register Mutex into Dual Boot instance
  #ifdef __MICODB_ENABLE_Mutex__
    MicoDBRegisterMutex(dualboot, mutex, 0x5);
  #endif
  // Dual Boot Reconfiguration
  MicoDB_Reconfigure (dualboot, efb);
  return(0);
}
```

Revision History

Component Version	Description
1.0	Initial Release.

.

Trademarks

Lattice Semiconductor Corporation, L Lattice Semiconductor Corporation (logo), L (stylized), L (design), Lattice (design), LSC, CleanClock, Custom Movile Device, DiePlus, E^2CMOS, Extreme Performance, FlashBAK, FlexiClock, flexiFLASH, flexiMAC, flexiPCS, FreedomChip, GAL, GDX, Generic Array Logic, HDL Explorer, iCE Dice, iCE40, iCE65, iCEblink, iCEcable, iCEchip, iCEcube, iCEcube2, iCEman, iCEprog, iCEsab, iCEsocket, IPexpress, ISP, ispATE, ispClock, ispDOWNLOAD, ispGAL, ispGDS, ispGDX, ispGDX2, ispGDXV, ispGENERATOR, ispJTAG, ispLEVER, ispLeverCORE, ispLSI, ispMACH, ispPAC, ispTRACY, ispTURBO, ispVIRTUAL MACHINE, ispVM, ispXP, ispXPGA, ispXPLD, Lattice Diamond, LatticeCORE, LatticeEC, LatticeECP, LatticeECP-DSP, LatticeECP2, LatticeECP2M, LatticeECP3, LatticeECP4, LatticeMico, LatticeMico8, LatticeMico32, LatticeSC, LatticeSCM, LatticeXP, LatticeXP2, MACH, MachXO, MachXO2, MACO, mobileFPGA, ORCA, PAC, PAC-Designer, PAL, Performance Analyst, Platform Manager, ProcessorPM, PURESPEED, Reveal, SiliconBlue, Silicon Forest, Speedlocked, Speed Locking, SuperBIG, SuperCOOL, SuperFAST, SuperWIDE, sysCLOCK, sysCONFIG, sysDSP, sysHSI, sysI/O, sysMEM, The Simple Machine for Complex Design, TraceID, TransFR, UltraMOS, and specific product designations are either registered trademarks or trademarks of Lattice Semiconductor Corporation or its subsidiaries in the United States and/or other countries. ISP, Bringing the Best Together, and More of the Best are service marks of Lattice Semiconductor Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.