Application Note, V1.1, Feb. 2002

AP3261

Quick Start with Hints on Using TC11IB

TC11IB

Microcontrollers

Never stop thinking.

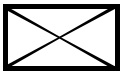
Revision History: 2002-02 V1.1

Previous Version:	V1.0
Page	Subjects (major changes since last revision)
7	Table 1: added ComDRAM_CLC entry
10	New section regarding eDRAM current
14	Addition discussion on back-to-back write to S_FPI and access to F_FPI or LMB

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com



1 Purpose of this document

This document is aimed to guide users of TC11IB to get a quick start to work with the Infineon TC11IB, Step BB or later (unless otherwise stated) device. It will also provide some application hints on using the device.

Some information provided here are available also on the TC11IB documentation. This document serves as a supplement to the TC11IB documents available - User's Manual, Data Sheet. Reference should always be made with these documents.

2 TC11IB - The Industrial Bus Controller

TC11IB is a high performance 32-bit byte-addressing controller running at 96 MHz CPU frequency that comes in P-BGA-388-2 package. Besides all the high-end features of the CPU TriCore and a PCP co-processor, TC11IB comes packed with peripherals like PCI, Ethernet, General Purpose Timers, Synchronous Serial Channel, Asynchronous Serial Channel, 16x50 (modem hardware or software flow control), Multimedia Card Interface (MMCI), watchdog timer and system timer.

3 Quick Start on TC11IB

3.1 Clock Circuitry

Reference should be made to the latest version of TC11IB Data Sheet.

Using the standard crystal circuitry as shown in the **Figure 1** below, recommended values for C1 and C2 is in the range 2 pF to 22 pF.

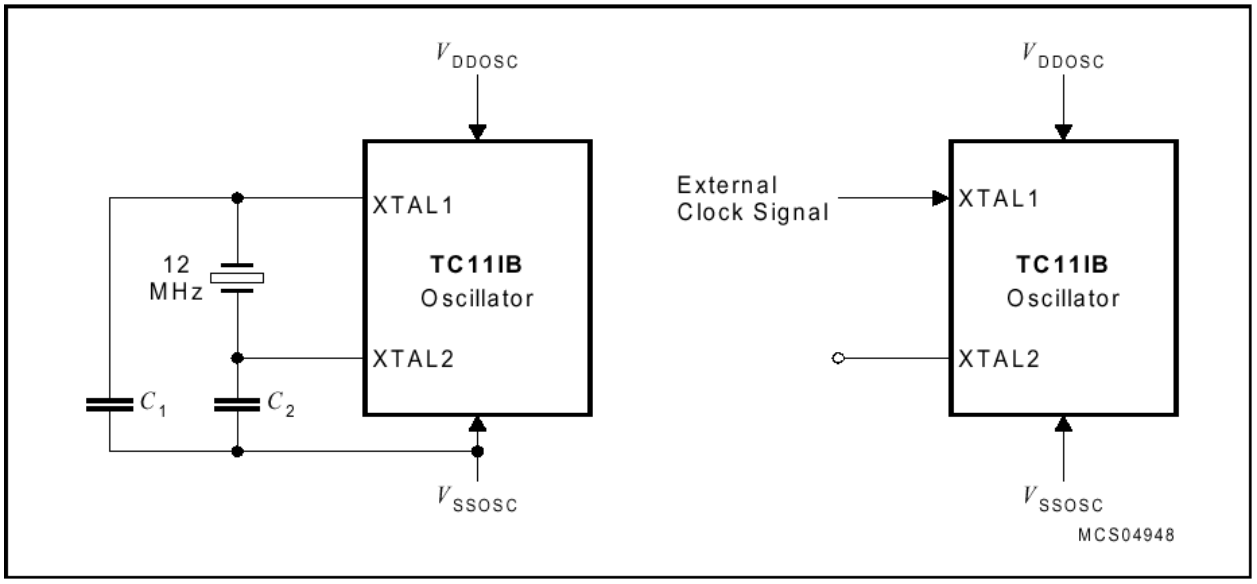


Figure 1: External Input Clock Circuitry

3.2 External Reset Circuitry

Consideration must be taken as to whether a manually applied reset on the hardware is to latch in (new) boot configuration and start another boot sequence or, simply to just reset the CPU and peripherals to run from default state. Two separate pins, #PORST# and #HRST#, respectively, are available for these purposes.

3.3 Boot Modes

The levels applied to pins #OCDSE#, #BRKIN#, CFG[3:0] when reset goes inactive determines the type of boot. TC11IB can start directly in core scratchpad memory, from external memory, go to halt in debug mode, or boot from one of the various modes in BOOT ROM. It is also possible to call the chip to deep sleep immediately.

Refer to **AP3230** titled 'Boot ROM Code TC11IB' for details.

3.4 Signal Reflection

TC11IB operates at high frequency and several pins will contain high frequency signals. At high frequencies, characteristic impedance of paths, lines become significant. Any impedance mismatch along the path may result in signal reflection, which is undesirable.

Special consideration should be made during the system hardware development stage to add terminating buffer resistors (and capacitors) along high frequency signal paths. Examples of high frequency signals from TC11IB are EBUCLK, ACLK, A0-A23, and AD0-AD31.

3.5 Watchdog Timer

The WDT in TC11IB automatically starts running after a reset. It should therefore be ensured that the WDT is refreshed before it times out. After reset, the WDT is in time-out mode. Setting ENDINIT to '1' enters normal mode. The default time-out period after reset is 682.67 μs, and can be increased up to 11.18 s in the normal mode. Alternatively, the WDT can be disabled, and this should be done in the beginning before all other initializations to ensure the watchdog does not time out.

To disable or refresh the WDT, it is required to follow a sequence to clear the ENDINIT bit in register WDTCON0 and at the end to set the ENDINIT bit again. After reset, the ENDINIT bit is cleared so the clear sequence is not required in the first instance. The following is a sample code in assembly of the full sequence.

```
;clear ENDINIT
mov d6, #-255     ;password access
ld.w d15, WDTCON0
and16 d15, d6
ld.w d7, WDTCON1
or d15, d15, #240
and d7, d7, #12
or16 d15, d7
st.w WDTCON0, d15
mov d7, #-16      ;modify access
and16 d15, d7
or d15, d15, #2
st.w WDTCON0, d15

mov   d7, #0x8    ; disable watchdog
st.w  WDTCON1, d7

;enable (set) EndInit protect
mov   d6, #-255   ; password access
ld.w  d15, WDTCON0
and16 d15, d6
ld.w  d7, WDTCON1
or    d15, d15, #240
and   d7, d7, #12
or16  d15, d7
st.w  WDTCON0, d15
mov   d7, #-16    ; modify access
and16 d15, d7
or    d15, d15, #3
st.w  WDTCON0, d15
```

Refer to **AP3219** for more information on how to use the WDT and **AP3221** to learn more about the ENDINIT bit and an example sequence in C.

3.5.1 ENDINIT-protected registers

This section highlights registers, which are ENDINIT-protected. **Table 1** is a consolidation of such registers of TC11IB.

Find more details about these registers in the TC11IB documents.

Table 1: TC11IB write ENDINIT-protected registers

Name	Absolute Address	Reset Value
ISP	F7E1FE28H	00000100H
BTV	F7E1FE24H	A0000100H
BIV	F7E1FE20H	00000000H
PCP_CS	F0003F10H	00000000H
PCP_CLC	F0003F00H	00000000H
EBU_CLC	F8000000H	00000000H
PCI_CLC	F0400000H	00000001H
ComDRAM_CLC	F0180000H	00000000H
MMCI_CLC	F0000B00H	00000000H
SSC_CLC	F0000A00H	00000002H
16x50_CLC	F0000900H	00000002H
ASC_CLC	F0000800H	00000002H
GPTU1_CLC	F0000700H	00000002H
GPTU0_CLC	F0000600H	00000002H
STM_CLC	F0000300H	00000000H
FFI_CON	F0000058H	00000000H
GSCON	F0000048H	00000000H
PLL_CLC	F0000040H	000F01XXH
PMG_CON	F0000030H	00000001H
WDT_CON1	F0000024H	00000000H
RSTREQ	F0000010H	00000000H

3.6 Professional Tools

TriCore has a complete tool suite for system development and debugging, such as compiler, assembler, debugger, and a configurable PCB referred to as 'Triboard'.

Development software is available from among others, Tasking, Green Hills and Hitex.
The TC11IB Triboard provides among others, ASC transceiver and DB9 connector, 16x50 transceiver, serial SPI-compatible EEPROM connected via SSC, two pairs of chip-selectable 4 Banks x 1 M x 16 bit SDRAM or two pairs of chip-selectable 4 Banks x 4 M x 16 bit SDRAM, and a pair of 1 M x 16 bit Burst Flash.

4 Application Hints

4.1 Signal Over-/Under-shoot, Ringing or Reflection Effects

Signal overshoot, undershoot due to ringing or reflection effects may be observed on some pins of certain modules in certain mode.
Some examples of these pins where *high frequency* toggling signals are observed to have overshoot and undershoot are:

1. SHIFT CLK pin of the ASC module in the synchronous mode at higher baud rates.
2. SSC pins of the SSC module at higher baud rates.
3. M16x50 TXD pin at higher baud rates.

To overcome such effects for the connecting peripheral, buffers can be used in the path of these pins. Out of the many possible solutions, here are some examples for 2 cases.

ASC synchronous mode, SHIFT CLK
Choose a transceiver that can buffer out the noise. In this 6 Mbaud example, MAX3227 is used to 'convert' the pin output (green, **Figure 2**) to a clean signal (pink) albeit the rise and fall time of signal is slower. It is recommended to refer to the MAX3227 data sheet to ensure the device characteristics and ratings match the actual application (e.g. T_IN maximum rating is -0.3 V to +6 V). An alternative is to use an R-C part before the transceiver, such as that described below for SSC.

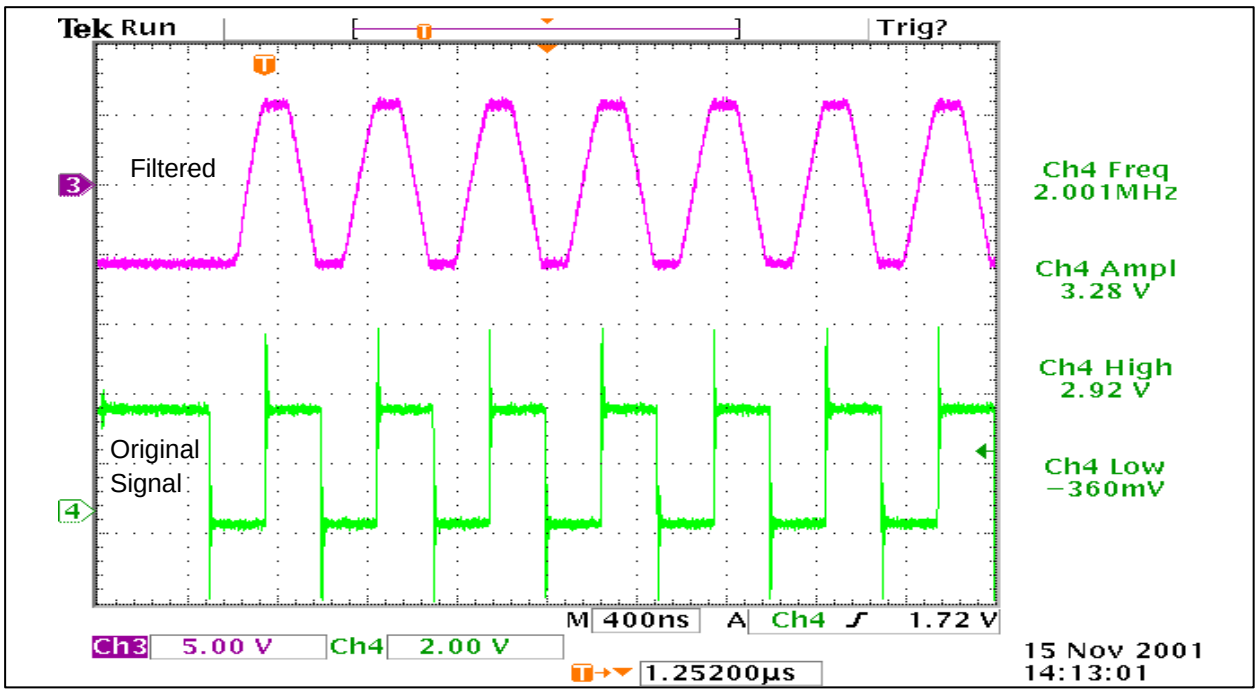


Figure 2: Filtered ASC signal with MAX3227

SSC master, MTSR

An R-C part helps, such as R=220 Ω, C=10 pF. **Figure 3** shows the MTSR output at 24 Mbaud (green) filtered to obtain a 'cleaner' signal (pink).

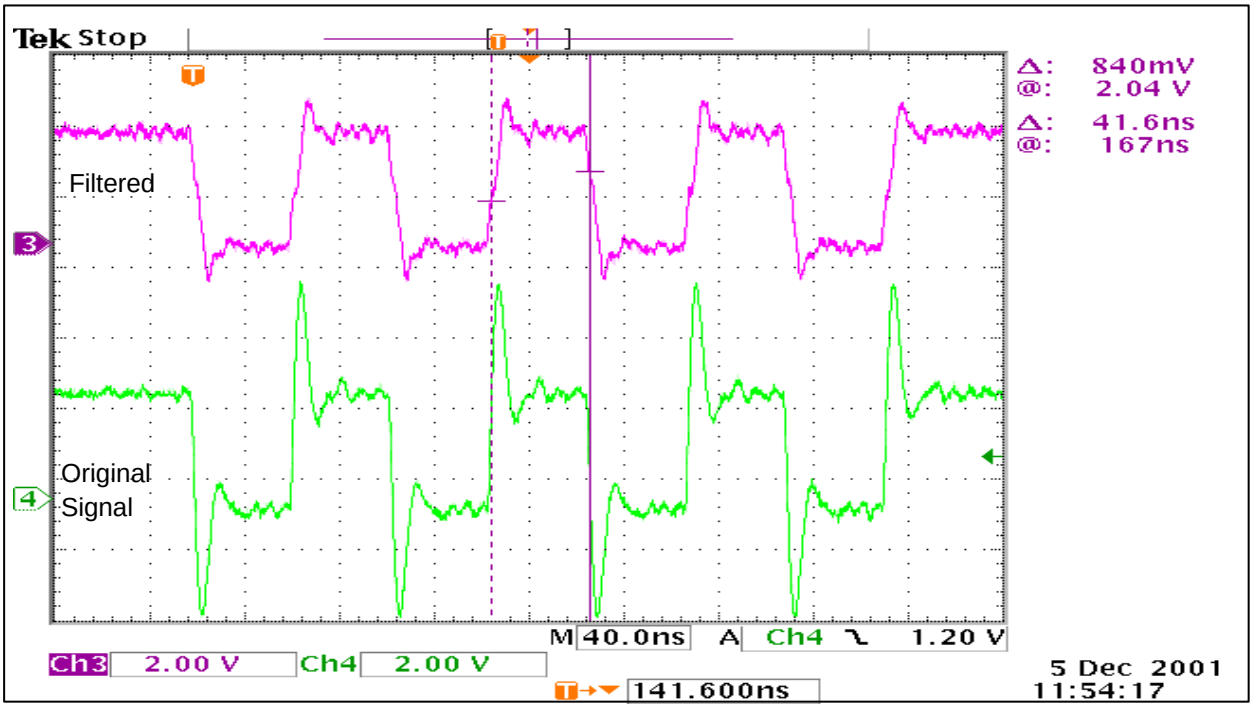


Figure 3: Filtered SSC signal with an R-C part

4.2 Power Down eDRAM Current

eDRAM power down current can be greatly reduced (less than 2 mA in deep sleep mode) if during #PORST# active (low), the following is true of the configuration pins:
TESTMODE = 1
CFG0 = 1
CFG1 = 1
CFG2 = 0
CFG3 = 0
Thereafter, these inputs should be set to levels according to the desired operational mode, which must be switched before #PORST# goes inactive.

One implementation is to manually switch the applied levels via a switch array while #PORST# is active. This allows flexibility to select among the various boot modes and is very useful on test boards.
Alternatively, where the boot mode is fixed on hardware, an INVERTER can be used with the RESET trigger (active LOW) as input (**Figure 4**). The inverted output is connected to the configuration pins that need be switched from '1' to '0'. This inverted output is inverted again to give the actual #PORST# signal to the TC11IB device and board. Configuration pins that need be switched from '0' to '1' can use the RESET trigger directly. INVERTER *U2A* is necessary to ensure the configuration pins are set for boot mode entry before #PORST# goes inactive. Important to ensure is the level at outputs of the INVERTER must match the DC specifications (V_{IL}, V_{IH}, I_{PU}, I_{PD} etc.), as on the TC11IB data sheet. User should confirm the implementation on own board as routing etc., may affect the intended logic.

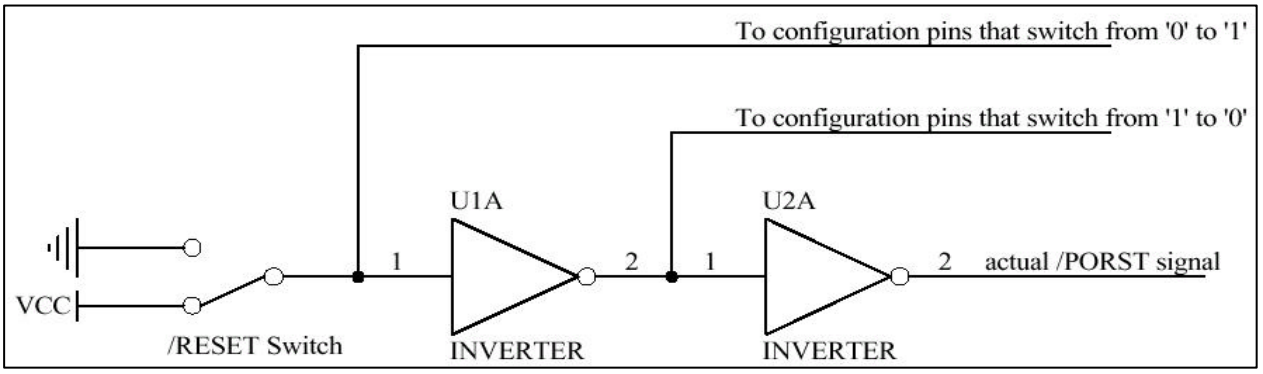


Figure 4: An example Implementation for Lower eDRAM Current

4.3 Back-to-back Write on the S_FPI, followed by Access to F_FPI or LMB

In TC11IB, TriCore itself is located on LMB bus, while the peripherals are connected to one of two FPI buses based on their speed due to the large number of peripherals on TC11IB. The 32-bit S_FPI and F_FPI bus are connected together via a bi-directional FFI-Bridge. The S_FPI bus runs at 48 MHz and connects other standard peripherals e.g. ASC, SSC, SCU with Watchdog Timer. The F_FPI bus runs at 96 MHz where most of the high performance peripheral e.g. ComDRAM, PCI-FPI, LFI-Bridge are connected. The LFI-Bridge interfaces between the F_FPI bus and the LMB bus to support bi-directional transactions.

These on-chip buses are interconnected as shown in **Figure 5**.
As shown in the figure, if any master of LMB bus e.g. CPU wants to access any peripheral on S_FPI bus, it has to go from LMB bus to F_FPI bus via LFI-Bridge, then to S_FPI bus via FFI-Bridge.

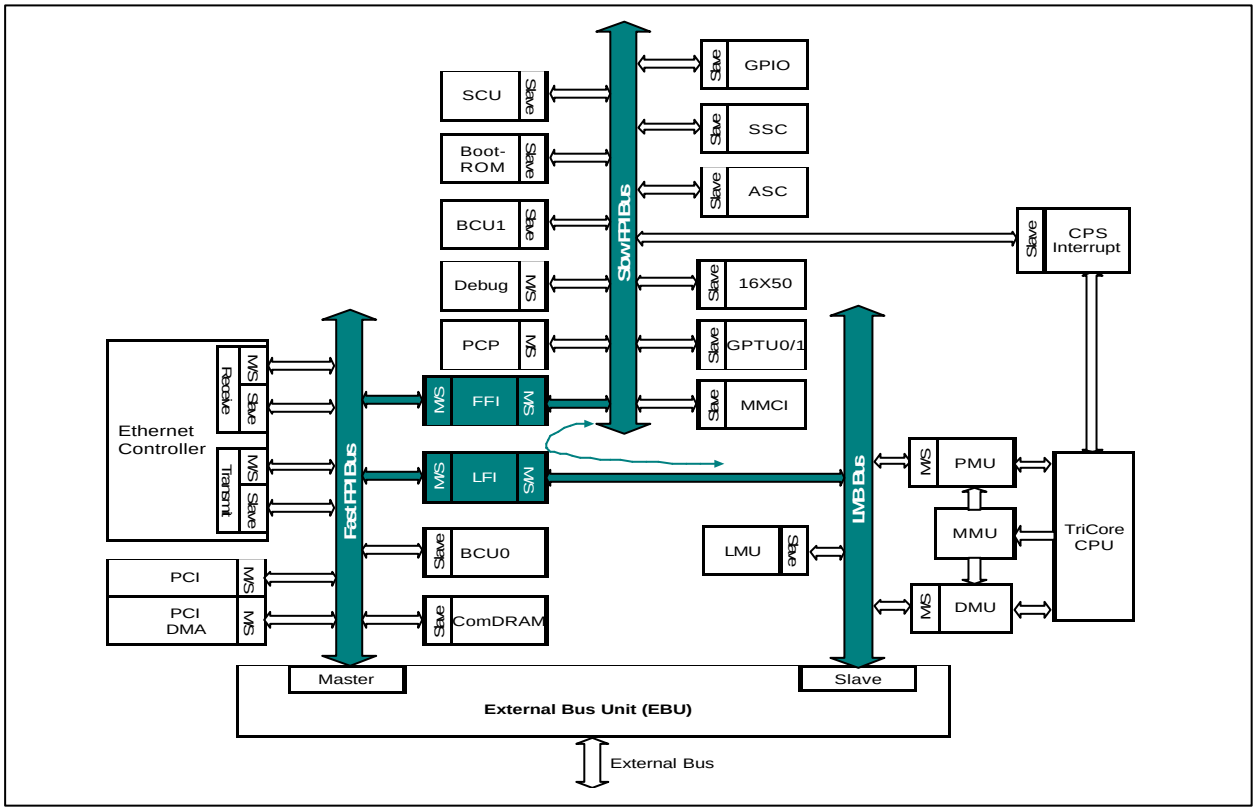


Figure 5: Block Diagram of Internal Bus Interconnections

Figure 6 shows a simplified timing diagram of two back-to-back LMB write operations. The first write operation is to a peripheral on S_FPI bus. The immediate second write operation is to a peripheral on the faster F_FPI. In normal operation, the results of two such write operations should not affect each other because the operations on F_FPI peripherals and S_FPI peripherals are independent. But if the second write operation depends on the result of the first write operation, then the second write operation will become invalid. As shown by the figure, the second write is 'done' even before the first write is ready. A classic example of such operations is when writing to EndInit-protected registers of peripherals on the F_FPI side.

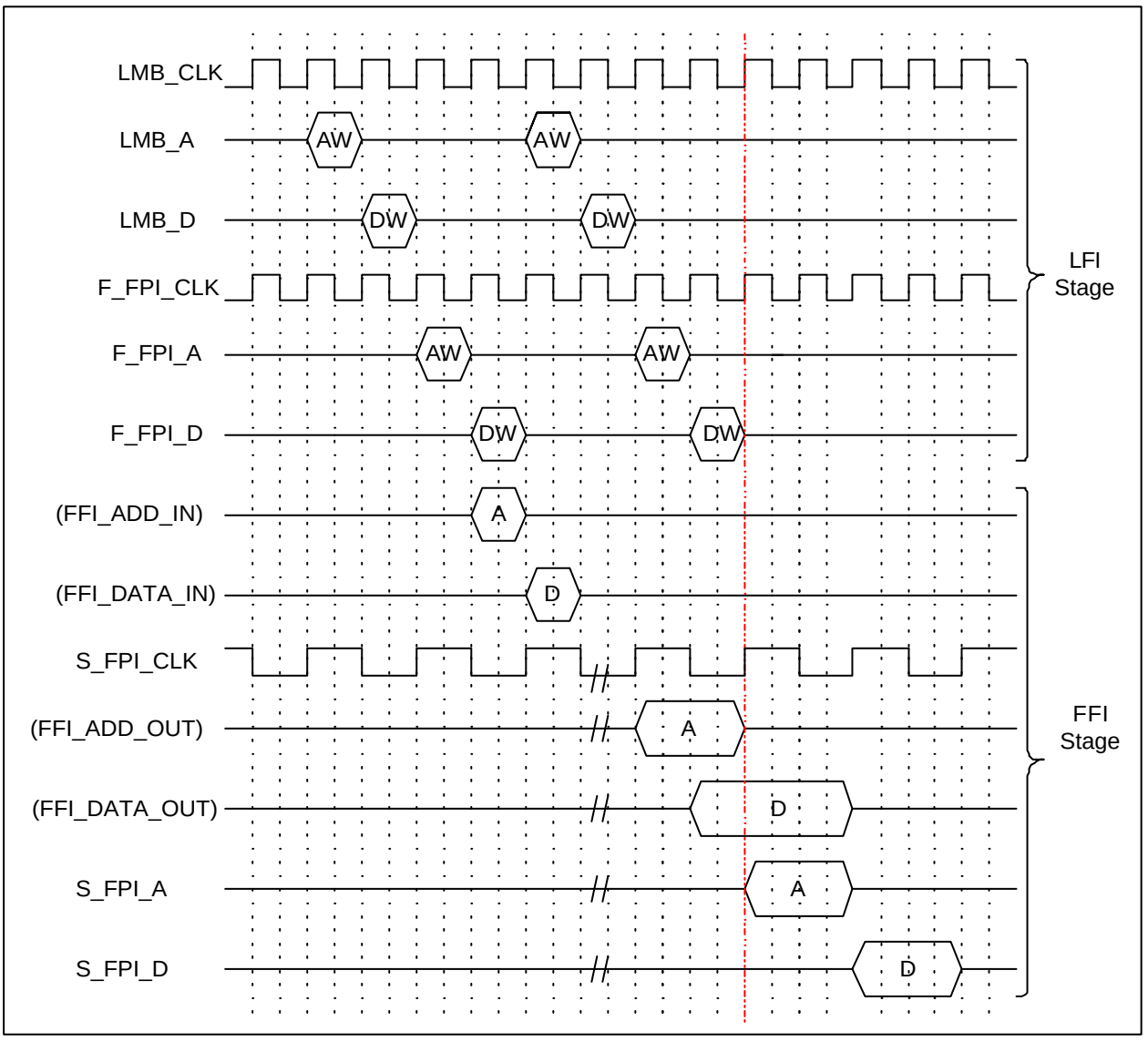


Figure 6: LMB Write to S_FPI, followed by Write to F_FPI

Here is more elaboration on such critical EndInit case, with a workaround.

The EndInit feature consists of an ENDINIT bit incorporated in the Watchdog Timer control register, WDT_CON0, which located at System Control Unit (SCU) hung on S_FPI Bus. A system-wide line is connected to this bit. Registers protected via EndInit use the state of this line to determine whether or not writes are enabled. Writes are only enabled if ENDINIT = 0 and Supervisor Mode is active. Write attempts if this condition is not true will cause a bus error and the register contents will not be modified.
To complete such operations successfully, the sequence should be as **Table 2**.

Table 2: Sequence of Using EndInit

1. ENDINIT to 0	WDT_Passwd(); //unlock WDT
	WDT_Modify(0x0, 0x1);//clear ENDINIT

2. Change one or many registers (Example: SSC_CLC)	//SSC_CLC is an example register SSC_CLC |= 0x00000001;//SSC clock enabled

3. ENDINIT to 1	WDT_Passwd(); //unlock WDT
	WDT_Modify(0x1,0x1); //set ENDINIT

If LMB issues two write operations back-to-back where the first write is to peripheral on S_FPI bus and the next write is for peripheral on F_FPI (or LMB bus), the updated result on S_FPI bus is slower than the result of immediate second write operation on F_FPI bus. This is critical since the second write will be effected only if result of first write is ready. This sequence happens in some cases, for example, an write operation to a register of peripherals on the F_FPI bus or LMB bus which is EndInit protected, e.g. EBU_CLC, PCI_CLC.

A simple workaround is to add a dummy read operation before the immediate second write operation after every write to S_FPI. The purpose is to insert wait time for the write operation to S_FPI. **Table 3** gives an example.

Table 3: Inserting dummy read between write to S_FPI and F_FPI

1. ENDINIT to 0	WDT_Passwd(); //unlock WDT
	WDT_Modify(0x0, 0x1);//clear ENDINIT

2. Dummy Read	WDT_Read(); //read WDT
3. Change one or many registers (Example: PCI_CLC)	//PCI_CLC is an example register PCI_CLC |= 0x00000001;//PCI clock enabled

4. ENDINIT to 1	WDT_Passwd(); //unlock WDT
	WDT_Modify(0x1,0x1); //set ENDINIT

The following procedure in C-Code is an example of dummy read function.

```
//Read WDTCON0, include this in your project
void WDT_Read(void)
    {
     #define WDTCON0 *((volatile unsigned int*) 0xF0000020)
     unsigned int dummyvalue;

     dummyvalue = WDTCON0; // load value from WDTCON0
     }
```

The above is focused on the case where second operation is a write access. Obviously, as long as the second operation is dependent on the completion of the first operation, a delay need be inserted whether the second operation is write or read access. An example latter case is to consider Ethernet:

```
;Check if Ethernet interrupt flag is set
_poll_eth_src:
read DRSRC        // example SRC register
if (DRSRC.SRR = 1) then
      write DRSRC, 0x4000     // clear the interrupt flag before read FIFO
      read DRISFIFO           // read the FIFO to check interrupt cause
      j _poll_eth_src         // check for more interrupt
else
      ...         //no more interrupt, continue with other task
end if
```

In this case, the DRISFIFO is a FIFO and will generate another interrupt (set bit SRR in DRSRC), if there are more content after being read. Since the read to DRISFIFO at the F_FPI is an instruction immediately after the write to clear SRR of DRSRC at the S_FPI, the next interrupt item indicated by DRISFIFO will be missed due to operation on F_FPI being faster than write to DRSRC.

5 Glossary

APXXXX	Application Note XXXX
ASC	Asynchronous serial channel
LMB	Local Memory Bus
F_FPI	Fast FPI
FIFO	First-In-First-Out
FPI	Flexible Peripheral Interface
S_FPI	Slow FPI
SRC	Service Request Control register
SRR	Service Request Flag of SRC
SSC	Synchronous serial channel
TC11IB	The integrated chip/device of interest
WDT	Watchdog timer

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results. Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG