NXP Semiconductors
Application Note

Document Number: AN5298
Rev. 0, 08/2016

S12Z Debug Module

Guidelines for using the S12Z Debug Module (S12ZDBG)

by: Peter Broderick

1 Introduction

The S12ZDBG module includes debugging features that were not available on S12/S12X predecessors. This application note provides guidelines for using these module features. A focus is put on using the profiling features.

- Section 1 introduces the module feature set.
- Section 2 discusses use of the state sequencer
- Section 3 discusses application of the profiling feature

This document describes the superset of debugging features offered by S12ZDBG V2 and V4, henceforth referred to as DBG module.

The S12ZDBG V3 (Debug Lite) includes a limited subset of the features that are not described in this document.

Table of Contents

1.1 Feature set

The following list is also provided in the DBG chapter of relevant device reference manuals. Items listed in **bold** are new features that are the focus of this document. Other features are similar to S12(X)DBG.

- Four Comparators. (A, B, C, and D)
  - Comparators A and C compare the full address bus and full 32-bit data bus
  - Comparators A and C feature a data bus mask register
  - Comparators B and D compare the full address bus only
  - Each comparator can monitor Program Counter (PC) addresses or addresses of data accesses
  - Each comparator can select either read or write access cycles
- Three comparator modes
  - Simple address/data comparator match mode
  - Inside address range mode
  - Outside address range match mode
- **Enhanced State sequencer control**
  - State transitions forced by comparator matches
  - State transitions forced by software write to TRIG
  - **Each channel (comparator) match can cause a transition to each state.**
  - **State transitions forced by an external event**
- The following types of breakpoints
  - CPU breakpoint entering active BDM on breakpoint (BDM)
  - CPU breakpoint executing SWI on breakpoint (SWI)
- Trace control
  - Tracing session triggered by state sequencer
  - Begin, End, and Mid alignment of tracing to trigger
  - Support for tracing through reset
  - **Timestamps**
- Trace modes
  - Normal: Change Of Flow (COF) PC information is stored
  - Loop1: same as Normal but inhibits consecutive duplicate source address entries
  - Detail: address and data for all read/write access cycles are stored
  - Pure PC: All program counter addresses are stored.
  - **Profiling Format: Compressed COF format used for profiling mode transmissions**
- **Profiling Mode**
  - **2 Pin (data and clock) profiling interface to output code flow information.**
  - **Internal access of compressed COF information from a profiling session**

1.2 DBG functional overview

The DBG module provides on-chip breakpoints and trace buffer with flexible triggering capability to allow non-intrusive debugging and profiling of application software. The DBG monitors internal Program Counter (PC) and system buses (see Figure 1). It can store system bus and PC information in the trace buffer and can be used to generate breakpoints to the CPU.

The comparators can be configured to monitor opcode addresses (effectively the PC address) or data accesses. Comparators can be configured during data accesses to mask out individual data bus bits. Comparators can be configured to monitor a range of addresses. When a match with a comparator register value occurs, the associated control logic can force the state sequencer to another state.

The state sequencer can transition freely between the states. State transitions can be configured to trigger tracing or generate breakpoints.

Independent of the comparators, state sequencer transitions can be forced by the external event input or by writing to the TRIG bit in the DBGC1 control register.

The trace buffer is visible through a 2-byte window in the register address map and can be read out using standard 16-bit word reads. This is typically handled over the device single pin (BKGD pin) interface to the debugger using (Background Debug Controller) BDC commands.

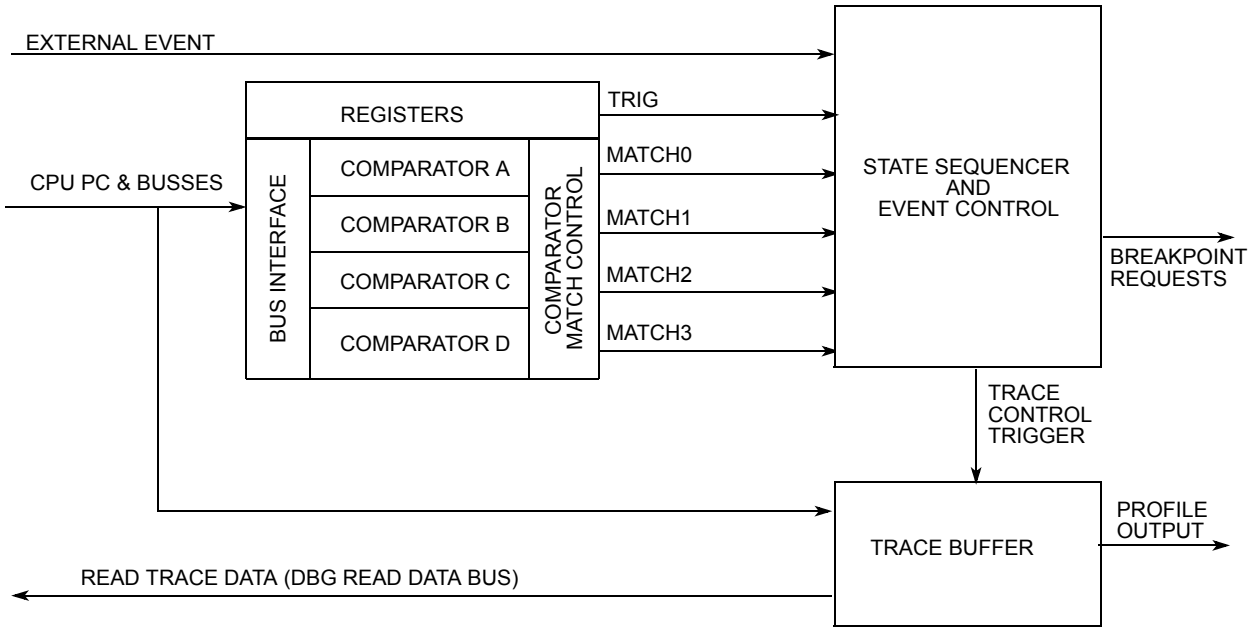

Figure 1. S12ZDBG block diagram

1.3 System interfacing

Typically the DBG module is used in conjunction with the BDC module, whereby the user configures the DBG module for a debugging session using the BDC in Background Debug Mode (BDM). In this mode the CPU is inactive whilst the user configures the DBG via the BDM interface. For more information please refer to the BDC chapter of a reference manual. Once configured, the DBG is armed and the device leaves BDM, returning control to the user program, which the DBG then monitors.

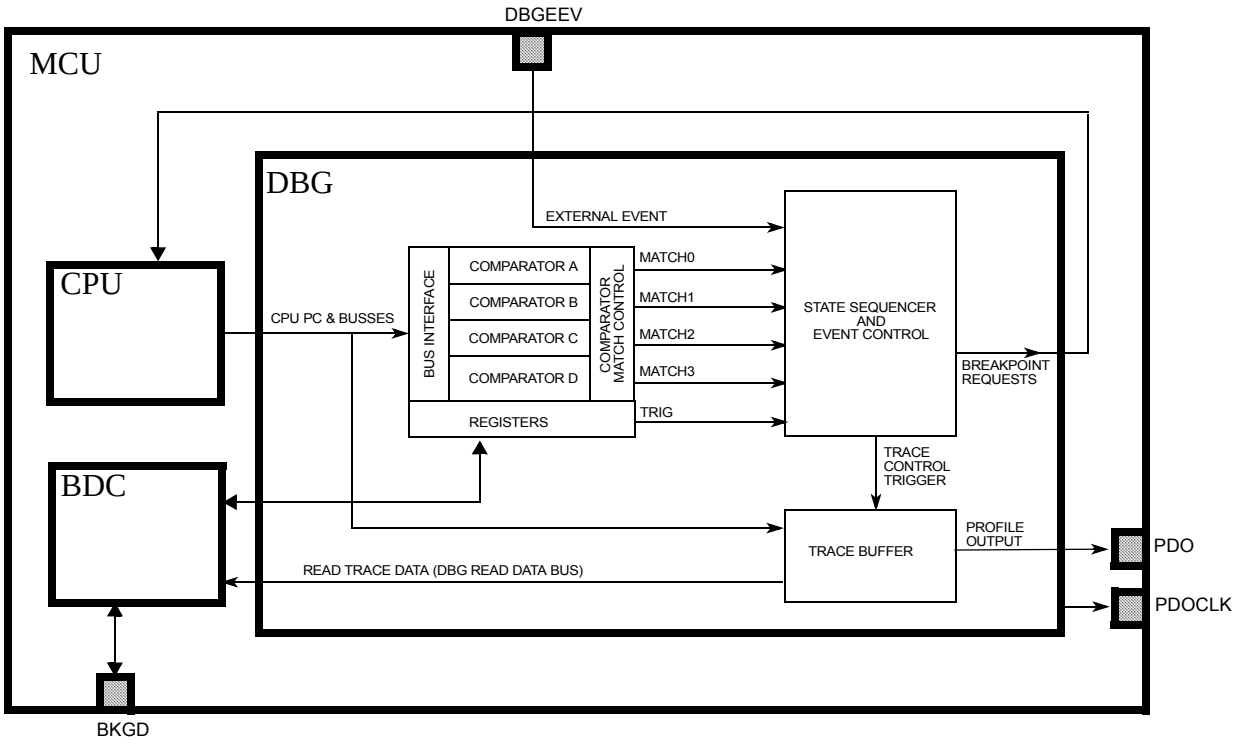

Figure 2. S12ZDBG interfacing

The DBG features three interface signals that are mapped to MCU device pins, in addition to the BKGD pin which is mapped to the BDC as shown in Figure 2.

The DBGEEV signal is input only and allows an external event to force a state sequencer transition, or trace buffer entry, or to gate trace buffer entries.

The Profiling Data Out (PDO) signal is output only and provides a serial, encoded data stream that can be used by external development tools to reconstruct the internal CPU code flow. During code profiling the device PDOCLK output is used as a clock signal to strobe the data transmitted on PDO.

At board level the profiling pins can use the same 6-pin connector typically used for the BKGD pin. The connector pin mapping shown in Figure 3 is supported by evaluation boards and development tools.

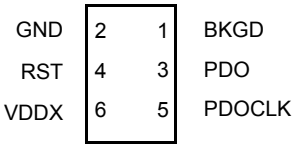

Figure 3. Standard debug connector pin mapping

2 Use of the state sequencer

During debugging it is often required to trigger a breakpoint or tracing session based upon a particular code sequence, as opposed to triggering simply at a certain PC value. The state sequencer can be configured to support a variety of sequences of comparator matches that can be used to trigger a breakpoint or tracing session. This section describes possible code sequence scenarios and corresponding DBG configurations.

Each state sequencer state 1, 2, 3 has a dedicated register, to select the targeted next state following comparator matches. The contents of the registers are identical, as shown in Figure 4.

		7	6	5	4	3	2	1	0
DBGSCR1	R	C3SC1	C3SC0	C2SC1	C2SC0	C1SC1	C1SC0	C0SC1	C0SC0
	W								
DBGSCR2	R	C3SC1	C3SC0	C2SC1	C2SC0	C1SC1	C1SC0	C0SC1	C0SC0
	W								
DBGSCR3	R	C3SC1	C3SC0	C2SC1	C2SC0	C1SC1	C1SC0	C0SC1	C0SC0
	W								

Figure 4. Debug state control registers (DBGSCRx)

Two bits are assigned to map each comparator match to a targeted next state. Thus it is possible to target different next states for different matches. External events can be mapped to Channel 3.

Table 1. DBGSCRx field descriptions

Field	Description
1–0 C0SC[1:0]	Channel 0 State Control. These bits select the targeted next state whilst in Statex following a match0.
3–2 C1SC[1:0]	Channel 1 State Control. These bits select the targeted next state whilst in Statex following a match1.
5–4 C2SC[1:0]	Channel 2 State Control. These bits select the targeted next state whilst in Statex following a match2.
7–6 C3SC[1:0]	Channel 3 State Control. If EEVE !=10, these bits select the targeted next state whilst in State1 following a match3. If EEVE = 10, these bits select the targeted next state whilst in State1 following an external event.

In each state, each match can force a transition to every other state or stay in the same state Table 2. The transition to final state can trigger a breakpoint or tracing session, depending on other DBG register settings.

Table 2. State1 match state sequencer transitions

CxSC[1:0]	DBGSCR1	DBGSCR2	DBGSCR3
00	Match has no effect	Match has no effect	Match has no effect
01	Match forces to State2	Match forces to State1	Match forces to State1
10	Match forces to State3	Match forces to State3	Match forces to State2
11	Match forces to Final State	Match forces to Final State	Match forces to Final State

In the case of simultaneous matches, the match on the higher channel number (3...0) has priority.

2.1 Scenario 1

In this and the following examples SCRx abbreviates DBGSCRx and Mx represents comparator MatchX.

In this scenario, a trigger is generated if a given sequence of 3 code events is executed. In each case the matches only cause a sequencer transition from within the defined state. If the matches do not occur in the given sequence (in this case M2 then M3 then M1) then the final state is not reached.

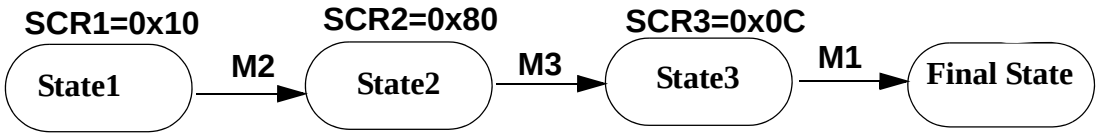

Figure 5. Scenario 1

2.2 Scenario 2

A trigger is generated if a given sequence of 2 code events is executed (Figure 6).

Figure 6. Scenario 2a

A trigger is generated if a given sequence of 2 code events is executed, whereby the first event is within a range (COMPA, COMPB configured for range mode). The comparator Match0 is mapped to the range of the memory map defined by COMPA and COMPB, such that any code event in that range generates a match. Note that M1 is disabled when COMPA and COMPB are configured for range mode. The second code event in the sequence is a single address mapped to Match3 (Figure 7).

Figure 7. Scenario 2b

A trigger is generated if a given sequence of 2 code events is executed, whereby either event is a range match and all comparators configured for range mode (Figure 8).

Figure 8. Scenario 2c

2.3 Scenario 3

A trigger is generated immediately when any one of up to 4 given events occurs.

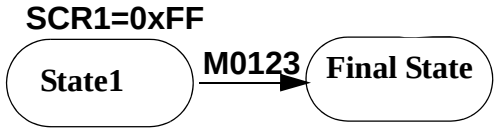

Figure 9. Scenario 3

2.4 Scenario 4

Trigger if a sequence of 2 events occurs in an incorrect order. In the application code event A (M0) must be followed by event B (M2) and M2 must be followed by M0. Two consecutive occurrences of M0 without an intermediate M2 are unexpected and cause a trigger. Similarly 2 consecutive occurrences of M2 without an intermediate M0 cause a trigger.

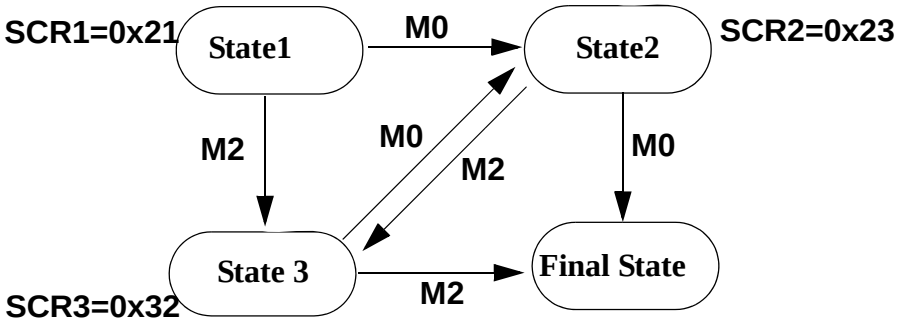

Figure 10. Scenario 4

Only 2 channels are used, so range comparisons can be included for both channel0 and channel2.

2.5 Scenario 5

Trigger when following Match2, Match0 precedes Match1. Thus if the expected execution flow is M2-> M1-> M0 but the actual execution flow is M2->M0 then final state is entered.

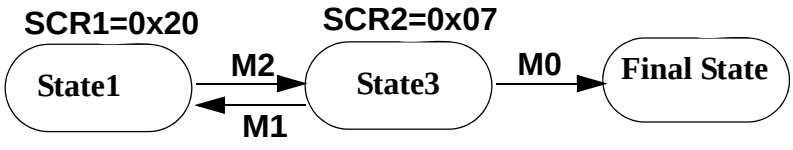

Figure 11. Scenario 5

2.6 Scenario 6

Trigger when Match2 occurs twice in succession before any other match occurs. Thus if an unexpected code loop occurs, then final state is entered.

Figure 12. Scenario 6

2.7 Scenario 7

Trigger when a series of 3 events is executed out of order. Specifying the event order as M1, M2, M0 to run in loops (M1->M2->M0->M1->M2->M0->M1->M2->M0). Any deviation from that order causes a transition to final state.

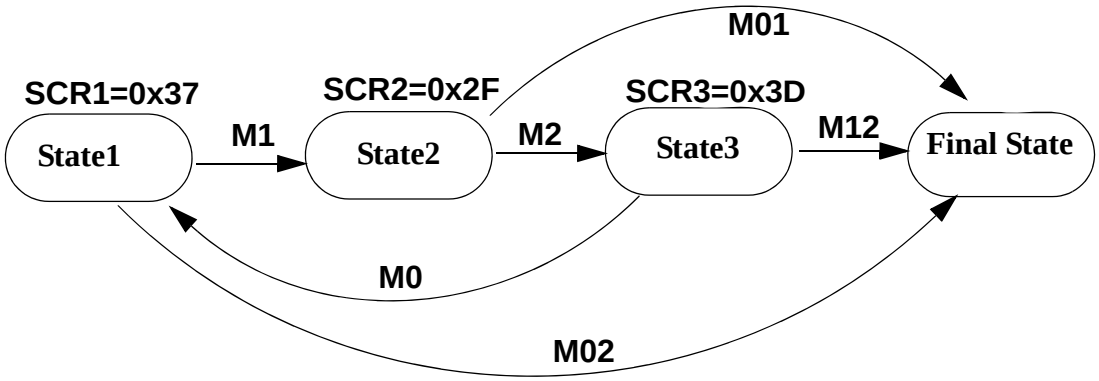

Figure 13. Scenario 7

2.8 Scenario 8

Trigger when an event at M3 follows either M2, M1 or M0.

Figure 14. Scenario 8

2.9 Scenario 9

Trigger when an event (M0, M1 or M2) is repeated before event (M3) occurs.

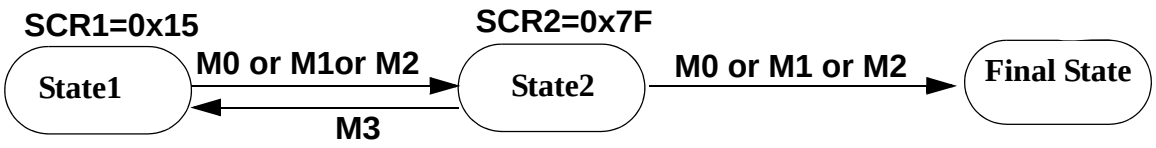

Figure 15. Scenario 9

3 Application of the profiling feature

During code profiling the MCU transmits encoded Change Of Flow (COF) information on the PDO pin and provides an associated reference clock on the PDOCLK pin. The internal trace buffer is used to store the information temporarily before it is transmitted. When the contents of a trace buffer line have been transmitted, then that line becomes available for storing new COF information. The DBG employs rollover whereby the oldest, transmitted lines are overwritten with new COF information. Thus the profiling session length is not limited by the size of the internal data buffer. This is discussed in Section 3.1

Independent of the profiling pin interface, the profiling data stored to the trace buffer can be read out in the usual manner using the BDC when the debug session ends. The profiling format contains COF information similar to the Normal trace mode, but in a highly compressed format. Thus the profiling configuration can be used to increase the trace depth. This is discussed in Section 3.2

3.1 Profiling using the PDO and PDOCLK pins

The code profiling data output pin PDO is typically mapped to a device pin that can also be used as GPIO in an application. If profiling is required and all pins are required in the application, it is recommended to use the device pin for a simple output function in the application, without feedback to the chip. In this way the application can still be profiled, since the pin has no effect on code flow.

The PDO provides a simple bit stream that must be strobed at both of the profiling clock edges whilst profiling. The first PDOCLK edge is used to sample the first data bit on PDO.

The external development tool activates the profiling output by setting the PROFILE and PDOE bits and then arming the DBG module. The first bit of the profiling bit stream is valid at the first edge of the profiling clock. No start bit is provided. The external development tool must detect this first edge. The end of profiling is indicated by the INFO byte bits TERM and TBOVF, and the DBGSR bit PTACT.

Figure 16 shows the profiling clock, PDOCLK, whose edges are offset from the bus clock, to ease setup and hold time requirements relative to PDO.

The development tool must first capture the PDO data and then decode. The possible trace buffer line formats whilst profiling are shown in Table 3. When profiling starts, the first line uses the PTS format. The transmission begins with bit0 of byte0 of this PTS entry. The development tool uses the INFO byte to

recognize the line format. In the case of the first line, it can be used to confirm the PTS format, and, if valid, the development tool can then decode the profiling start address from the following bytes.

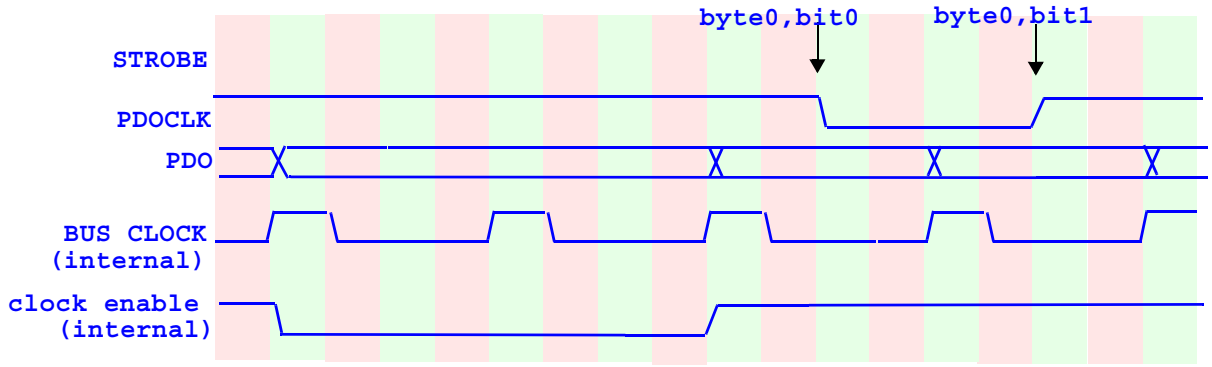

Figure 16. PDO profiling clock and start of transmission

3.1.1 Profiling format

The transmitted profiling data format is defined by the format in which the data is stored in the trace buffer before transmission. Each trace buffer line consists of 8 bytes. Each line can be used to store between 4 and 8 bytes of profiling data, using one of the formats shown in Table 3.

The profiling format is highly compressed because each direct COF is stored to a single bit in the trace buffer. Up to 4 bytes of each line are dedicated to branch COFs. In practical use cases COF events occur several cycles apart, so the PDO transmission rate exceeds the rate of filling the trace buffer. This allows profiling of an unlimited length of a typical running application.

The line format encoded in the INFO byte indicates the number of bytes of trace information that the line contains, which is an essential reference for decoding the following information. Thus for a PTS format, 4 bytes in total are transmitted before proceeding to the next line; for the PTHF format 6 bytes are transmitted; otherwise all 8 bytes are transmitted. The gray fields indicate not-transmitted bytes.

Table 3. Profiling trace buffer line format

Format	8-Byte Wide Trace Buffer Line							
	7	6	5	4	3	2	1	0
PTS					PC Start Address			INFO
PTIB	Indirect	Indirect	Indirect	Direct	Direct	Direct	Direct	INFO
PTHF			0	Direct	Direct	Direct	Direct	INFO
PTVB	Timestamp	Timestamp	Vector	Direct	Direct	Direct	Direct	INFO
PTW	Timestamp	Timestamp	0	Direct	Direct	Direct	Direct	INFO

After the initial PTS entry, containing the PC start address, an internal pointer increments and the DBG starts to fill the next line with COF information from the subsequent code flow. This continues until the 4-byte direct COF field is full or an indirect COF or interrupt occurs. Then the INFO byte and, if needed, indirect COF or vector information are entered on that line, and the pointer increments to the next line.

Direct COFs have destination addresses that are known at compile time and do not change during run time. Thus a single bit is sufficient to indicate if a direct COF is taken or not (see Section 3.1.1.1).

The PTHF format is used if the direct COF field is filled before an indirect COF or interrupt occurs. The PTIB format is used if an indirect COF occurs before the direct COF field is filled. The PTVB format is used if an interrupt occurs before the direct COF field is filled.

The destination addresses of indirect COFs and interrupts are determined during runtime, they cannot be determined during code compilation. Thus their full destination address is recorded for reconstruction of code flow. Indirect COFs force a full line PTIB format entry, requiring 3-bytes for the full 24-bit destination address. Interrupts force a full line PTVB format entry, whereby vectors are stored as a single byte and a 16-bit timestamp value is stored simultaneously to indicate the number of core clock cycles relative to the previous COF. The device vectors use address[8:0] whereby address[1:0] are constant zero for vectors. The value stored to the PTVB vector byte is equivalent to Vector Address[8:1].

If a timestamp overflow occurs, indicating a 65536 core clock cycles without COF, then an entry is made with the TSOVF bit set and profiling continues.

If the trace buffer contains 64 lines pending transmission then a trace buffer overflow occurs, a final entry is made with the TBOVF bit set. Then profiling data storage is terminated and the DBG is disarmed. However, profiling transmission continues until all lines have been sent.

Whenever the DBG is disarmed during profiling, a final entry is made with the TERM bit set to indicate the final entry. When a final entry is made then by default the PTW line format is used, except if a COF occurs in the same cycle in which case the corresponding PTIB/PTVB/PTHF format is used.

Development tools receive the INFO byte first, so can determine in advance the format of data it is about to receive. The INFO byte transmission starts when a line is complete. Whole bytes are always transmitted.

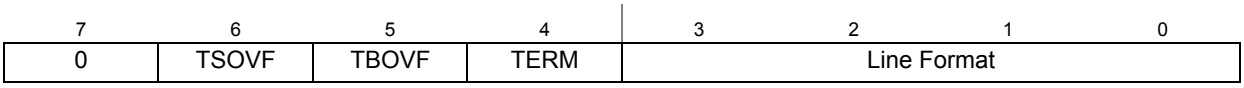

Figure 17. INFO byte encoding

Table 4. Profiling format encoding

INFO[3:0]	Line Format	Source	Description
0000	PTS	CPU	Initial CPU entry
0001	PTIB	CPU	Indexed jump with up to 31 direct COFs
0010	PTHF	CPU	31 direct COFs without indirect COF
0011	PTVB	CPU	Vector with up to 31 direct COFs
0111	PTW	CPU	Error (Error codes in INFO[7:4])
Others	Reserved	CPU	Reserved
INFO[7:4]	**Bit Name**		**Description**
INFO[7]	Reserved	CPU	Reserved
INFO[6]	TSOVF	CPU	Timestamp Overflow
INFO[5]	TBOVF	CPU	Trace Buffer Overflow
INFO[4]	TERM	CPU	Profiling terminated by disarming
Vector[7:0]	Vector[7:0]	CPU	Device Interrupt Vector Address [8:1]

3.1.1.1 Direct COF compression

Each branch COF is stored to the trace buffer as a single bit (0=branch not taken, 1=branch taken) until an indirect COF (indexed jump or return) or interrupt occurs. The branch COF entries are stored in the byte fields labeled “Direct”. These entries start at byte1[0] and continue through to byte4[7], or until an indirect COF occurs, whichever occurs sooner. The entries use a format whereby the left most asserted bit is always the stop bit, which indicates that the bit to its right is the first direct COF and byte1[0] is the last COF that occurred before the indirect COF. This is shown in Table 5, whereby the Bytes 4 to 1 of the trace buffer are shown for 3 different cases. The stop bit field for each line is shaded.

In line0, the left most asserted bit is Byte4[7]. This indicates that all remaining 31 bits in the 4-byte field contain valid direct COF information, whereby each 1 represents branch taken and each 0 represents branch not taken. The stop bit of line1 indicates that all 30 bits to it’s right are valid, after the 30th direct COF entry, an indirect COF or interrupt occurred. In this case the bit to the left of the stop bit is redundant. Line2 indicates that an indirect COF or interrupt occurred after 8 direct COF entries. All bits to the left of the stop bit are redundant.

Table 5. Profiling direct COF format

Line	Byte4								Byte3								Byte2								Byte1							
Line0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	1	1	0
Line1	0	1	1	0	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0
Line2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1

3.1.2 Profiling data transmission

Profiling transmission begins at the first falling edge of PDOCLK with trace buffer line0, byte0, bit0 (INFO byte, bit0) as shown in Figure 18. The first entry uses PTS format, so the INFO byte line format field, bits[3:0], contains 0x0. The following bytes are also transmitted starting with the least significant bit. Thus the first transmitted bit of the PC start address is PC[0]. Without exception, each clock edge corresponds to a data bit, thus transmission of each byte continues from the previous byte with the next clock edge. Similarly transmission of each (trace buffer) line continues from the previous line using the next clock edge, as shown in Figure 18. If a trace buffer line transmission completes before the next trace buffer line is ready, then PDOCLK stops clocking until the line is ready for transfer.

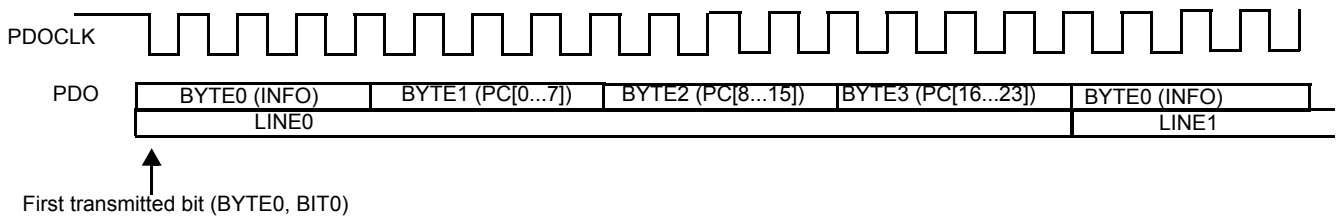

Figure 18. Profiling transmission start

Debugging at assembler level allows a better understanding of the code flow. The COF information is directly related to the assembler opcode addresses and vectors. The following examples show how the transmitted profiling data maps to CPU code flow.

3.1.2.1 Example 1 PTS followed by PTHF

Transmission starts with the INFO byte contents 0x00 indicating format PTS and continues with the initial PC value 0xFC00C4 in bytes[3:1] whereby the least significant bit is transmitted first. Then the contents of the next line are transmitted. This line uses format PTHF because 31 consecutive direct COFs occurred without any indirect COF, corresponding to the code flow given below. The first COF in the flow is the bit to the right of the stop bit. The sequence is shown in red in the example.

```
fc00c4 A476     LD D0,#6 ...      Code start address (PTS Contents)
fc00c6 44       DEC D0
fc00c7 F471     CMP D0,#1
fc00c9 247D     BCC (0xfc00c6) :  Branch taken 5 time(s) then not taken 111110
fc00cb 01       NOP
fc00cc 9400     LD D0,#0
fc00ce 34       INC D0
fc00cf F476     CMP D0,#6
fc00d1 257D     BCS (0xfc00ce) :  Branch taken 5 time(s) then not taken 111110
fc00d3 01       NOP
fc00d4 A473     LD D0,#3
fc00d6 34       INC D0
fc00d7 F476     CMP D0,#6
fc00d9 277D     BEQ (0xfc00d6) : Branch not taken.0
fc00db 01       NOP
fc00dc 9400     LD D0,#0
fc00de 34       INC D0
fc00df F478     CMP D0,#8
fc00e1 2B7D     BMI (0xfc00de) :  Branch taken 7 time(s) then not taken 11111110
fc00e3 01       NOP
fc00e4 9400     LD D0,#0
fc00e6 34       INC D0
fc00e7 F476     CMP D0,#6
fc00e9 267D     BNE (0xfc00e6) :  Branch taken 5 time(s) then not taken 111110
fc00eb A478     LD D0,#8
fc00ed 44       DEC D0
fc00ee F471     CMP D0,#1
fc00f0 2A7D     BPL (0xfc00eb) :  Branch taken 4 times 1111
```

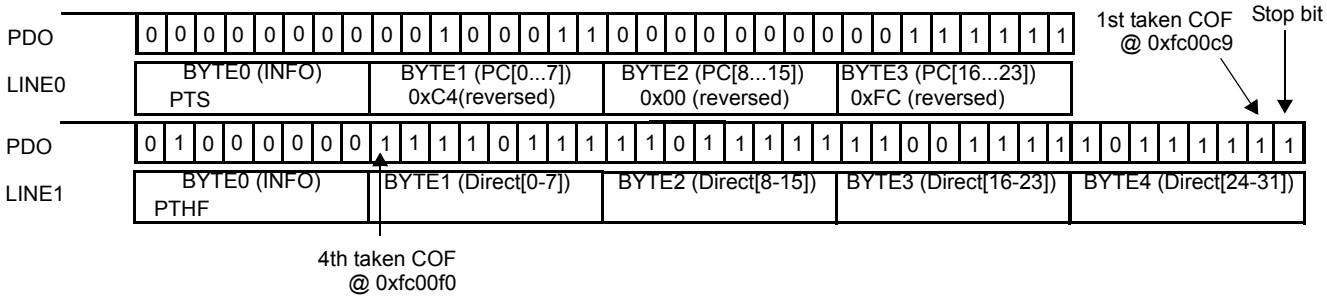

Figure 19. Transmission example 1: PTS followed by PTHF

3.1.2.2 Example 2 PTIB

The INFO byte contents 0x01 indicate format PTIB and continues with the direct COF field bytes. This line uses format PTIB because an indirect COF occurred, which requires a full destination address entry before all direct COF field bits can be used. The first COF in the flow is the bit to the right of the stop bit. The sequence is shown in red in the example.

```
fe002e 02B006C700       BRCLR.B (0xfe002e) Branch taken 5 times then not taken.111110
fe0033 BC06DB           CLR.B 1755
fe0036 A406CC           LD D0,1740
fe0039 9E40             TFR D0,D2
fe003b 5800BF           AND D2,#191
fe003e 780020           OR D2,#32
fe0041 9E04             TFR D2,D0
fe0043 C406CC           ST D0,1740
fe0046 BC06D0           CLR.B 1744
fe0049 1C7106DD         MOV.B #1,1757
fe004d BAFE00A0         JMP (0xfe00a0) Not Indexed, thus not conditional COF!
fe00a0 1B030011FF       LD S,#4607
fe00a5 BBFE0072         JSR (0xfe0072) Not Indexed! (Return Address fe00a9)
fe0072 B6FE0051         LD D6,16646225
fe0076 2714             BEQ (0xfe008a) Branch taken 1 time then not taken 10
fe008a 05               RTS (0) Indirect COF destination is fe00a9
```

The non-indexed jumps are not conditional because the destination is fixed in the code and the destination address determined by the compiler. Thus they are not classed as COF events and are not recorded.

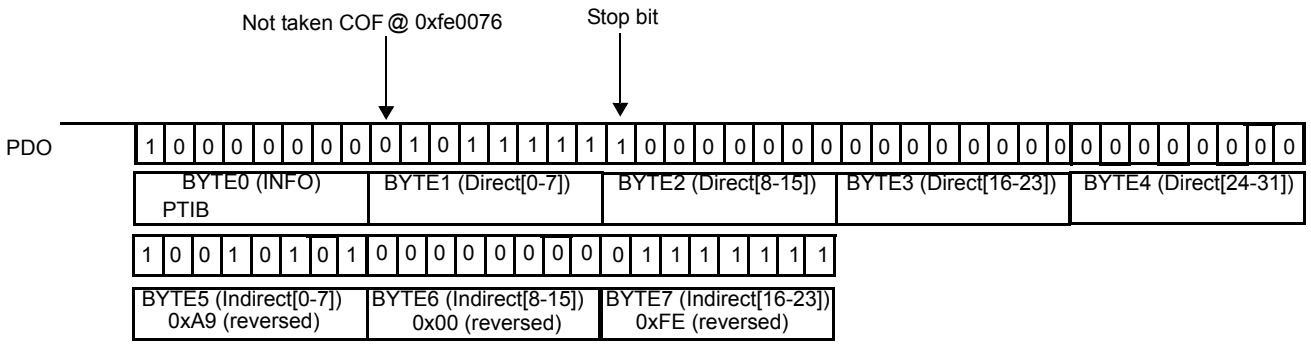

Figure 20. Transmission example 2: PTIB

3.1.2.3 Example 3 PTVB following PTHF

In this case the PTS is not shown, the example starts in the middle of a profiling session, with the transmission of a PTHF line format, where all 31 direct COF indicate a branch taken, which is typical of code loops. The redundant byte 5 is also transmitted.

```
fe00f3 9F101B INC.L 4123
fe00f6 A6101B LD D6,4123
fe00f9 E600030D40 CMP D6,#200000
fe00fe 2575 BCS (0xfe00f3) :  Branch taken 31 times 11111111111111111111111111111111
```

All COFs were taken, so the branch loop continues on the next trace buffer line. This following line uses the PTVB format because an interrupt occurred. The INFO byte contents 0x03 indicate format PTVB. The next 4 bytes contain direct COF fields. The stop bit is byte 2, bit 4.

```
fe00f3 9F101B INC.L 4123
fe00f6 A6101B LD D6,4123
fe00f9 E600030D40 CMP D6,#200000
fe00fe 2575 BCS (0xfe00f3) :  Branch taken 12 times 111111111111
```

The final direct COF entry was also taken, so the loop was still being executed when the interrupt occurred.

The vector byte contents of 0xC2 map directly to vector address[8:1]. Thus the actual vector is calculated by simply adding vector address[0], which always has a zero value. This gives the real vector 0x184. The timestamp count of 0x0A converted to decimal, 10, indicates that 10 core clock cycles occurred between the last COF entry and the interrupt.

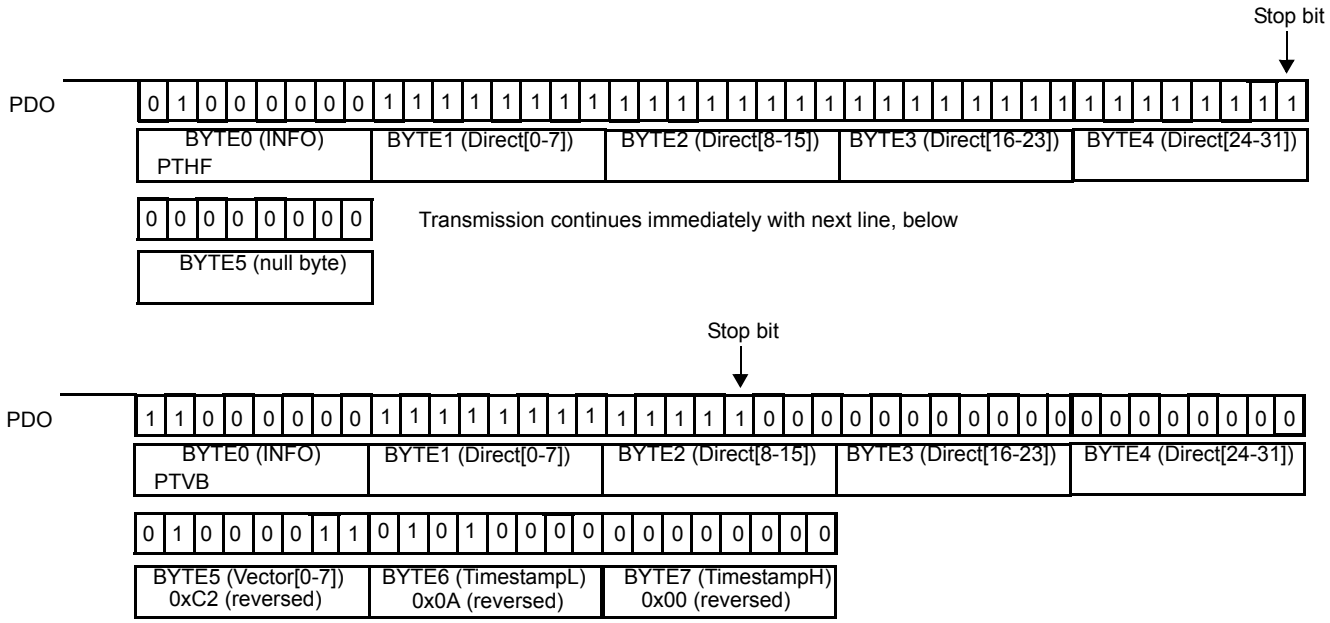

Figure 21. Transmission example 3: PTHF followed by PTVB

3.2 Tracing using the profiling format

The profiling configuration can be used as an additional trace mode, without the profiling pins. This offers a significantly increased trace depth in comparison to the other trace modes because many direct COFs can be stored to a single trace buffer line using the profiling format. However, in this case, the compressed format is more complicated to decode then the other trace modes Normal, Loop1, Detail and PurePC.

The DBG V2 does not feature the PREND bit. The use of the profiling format as a trace mode on DBG V2 is limited to debug sessions where no trace buffer rollover has occurred. This is indicated by the TBF status bit. If configured for end aligned profiling, a trigger to final state can be used to force the session to end before a rollover occurs, however this may not be appropriate in scenarios with unexpected code flow.

The DBG V4 features the PREND bit, which forces the profiling session to terminate when the trace buffer is full. This prevents a rollover from overwriting the PC start address stored in trace buffer line0. Thus the profiling format can generally be used as a trace mode on DBG V4 without forcing a session end.

It is recommended to use the BDC READ_DBGTB command to read the trace buffer because this is the most efficient method of reading the contents. It is important to note the byte order. The trace buffer read index starts at the lowest address, so the least significant 4 bytes are read first, then bytes 7:4. Development tools that read and transmit after each read list them then in the order:- 3,2,1,0,7,6,5,4.

Using other 16-bit BDC read commands, apart from being less efficient, also mix the byte order up more since they would return the bytes in the order 1,0,3,2,5,4,7,6, based on four 16-bit reads.

The Example 1 code would be stored to the trace buffer as shown in Table 6.

Table 6. Example of tracing using the profiling format

Format	8-Byte Wide Trace Buffer Line							
	7	6	5	4	3	2	1	0
PTS	0x00	0x00	0x00	0x00	0xFC	0x00	0xC4	0x00
PTHF	0x00	0x00	0x00	0xFD	0xF3	0xFB	0xEF	0x02

Reading the trace buffer using READ_DBGTB delivers the bytes in the order shown:-

```
0xFC00C400 0x00000000; Line0 bytes[3:0], bytes[7:4]
0xF3FBEF02 0x000000FD; Line1 bytes[3:0], bytes[7:4]
```

Note that although the shaded bytes are not transmitted on the PDO pin, they are always read out when reading from the trace buffer because the reads do not recognize redundant bytes.
The first line byte[0] value of 0x00 indicates PTS format, bytes[3:1] give the start PC address 0xFC00C4.
The second line is decoded as follows

```
0xF3FBEF02 : Byte0 (INFO) = 0x02 -> PTHF
```

Rearranging direct COF bytes[4:1]. 0x**F3FBEF**02 0x000000**FD**; -> 0xFDF3FBEF.

Mapping direct COF bits to actual code, starting with bit right of Stop bit. 0xFD = 0b1111.1101. Thus the first 5 direct COFs following the PC start address are taken, the next one is not. As the branch is a simple loop it is clear that the same branch was taken 5 times, then on the sixth time not taken.

```
fc00c6 44       DEC D0
fc00c7 F471     CMP D0,#1
fc00c9 247D     BCC (0xfc00c6) :  Branch taken 5 time(s) then not taken 111110
```

The PTVB format timestamp indicates the number of core clock cycles between the interrupt and the previous COF. Generally the interrupt return address reveals the position in code flow where the interrupt occurred, so the timestamp information is redundant. However, if the return address has been changed by stack manipulation then the timestamp information can indicate where in code flow the interrupt occurred. This may be insufficient to reconstruct code flow exactly, without a knowledge of the cycle count for each instruction between previous COF and interrupt. If this is the case then the debug session can be repeated using PurePC mode using the previous COF as a trigger.

4 References

1. MC9S12ZVM-Family Reference Manual, NXP Semiconductors 2012-2016.

How to Reach Us:

Home Page:

nxp.com

Web Support

nxp.com/support

Document Number: AN5298
Rev. 0
August 2, 2016