MTB Debugger

Introduction:

The idea of this project is to demonstrate a bootstrap implementation of Micro trace buffer peripheral provided in the ARM Cortex M0+ architecture using the KL25Z microcontroller. Using this peripheral to trace the entire call stack history of a program execution, transmitting it via an SPI bus to an external Host device, and logging it to this device. The project also aims to decipher the MTB trace information using the map file generated for the source code and providing a timeline of the code execution.

Literature study:

Microcontrollers using the Cortex-M0+ processor core include support for a CoreSight Micro Trace Buffer to provide program trace capabilities. The proper name for this function is the CoreSight Micro Trace Buffer for the Cortex-M0+ Processor. The simple program trace function creates instruction address change-of-flow data

packets in a user-defined region of the system RAM. Accordingly, the system RAM controller manages requests from two sources:

- AMBA-AHB reads and writes from the system bus

- Program trace packet writes from the processor

As part of the MTB functionality, there is a DWT (Data Watchpoint and Trace) module that allows the user to define watchpoint addresses, or optionally, an address and data value, that when triggered, can be used to start or stop the program trace recording.

Following block diagram is taken from the KL25Z refence manual (*CH 19, pg-300*).

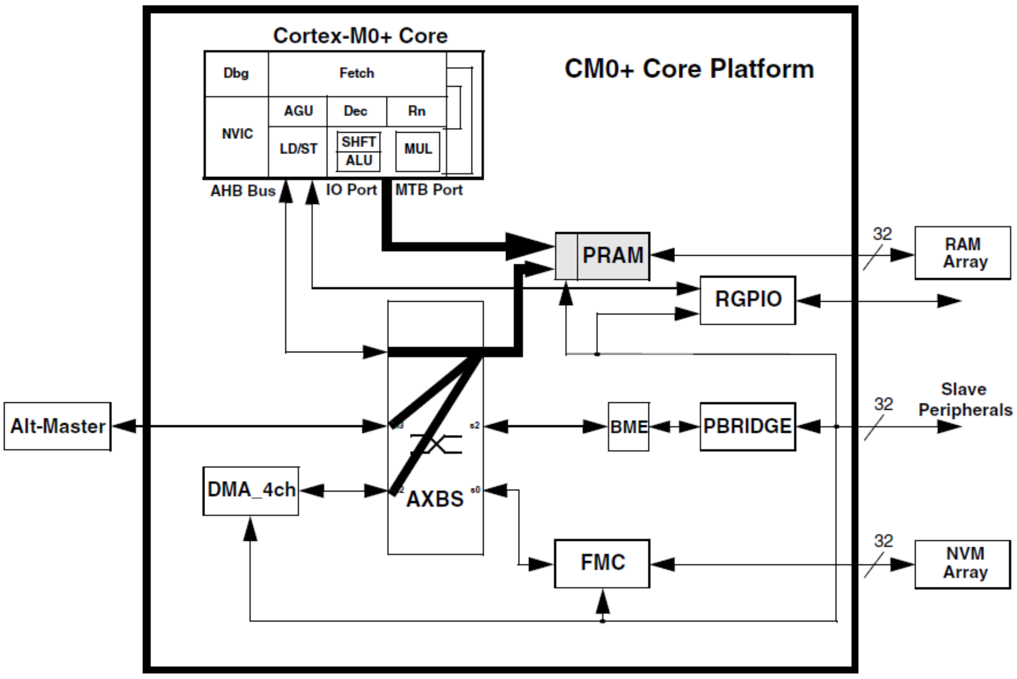


Figure 19-1. Generic Cortex-M0+ core platform block diagram

The logical paths from the crossbar master input ports to the PRAM controller are highlighted along with the private execution trace port from the processor core. The private MTB port signals the instruction address information needed for the 64-bit program trace packets written into the system RAM. The PRAM controller output interfaces to the attached RAM array. In this document, the PRAM controller is the MTB_RAM controller.

The ARM CoreSight Micro Trace Buffer documentation describes the trace packet received as -

"The execution trace packet consists of a pair of 32-bit words that the MTB generates when it detects the processor PC value changes non-sequentially. A non-sequential PC change can occur during branch instructions or during exception entry."

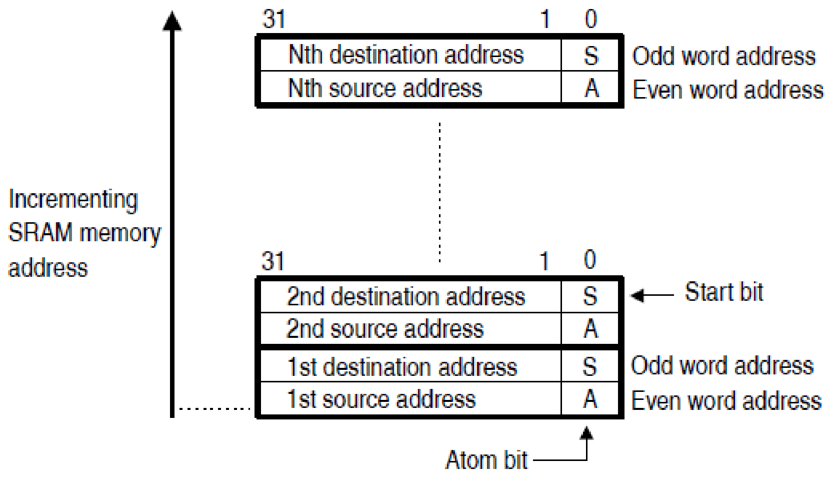
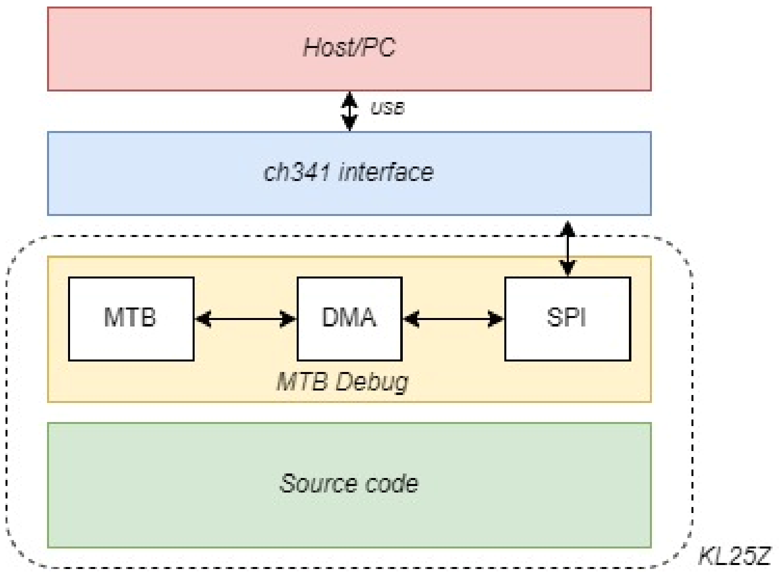


Figure 19-2. MTB execution trace storage format

Implementation ideology:

The following layer diagram shows the initial implementation approach -



Any code that needs to be debugged using this implementation includes a header file called mtb_debug.h. The library will be consisting primarily

1. MTB configurations
2. DMA configs and initialization
3. SPI peripheral initialization

The MTB configurations would be made in such a way that a macro defined in the source code to enable the debug functionality. These configurations will include the trace buffer size, location, and mode of the MTB trace. MTB_RAM will be used for this project.

The DMA is configured as a memory-to-memory transfer bus which will be used to transfer the Trace buffer data to the SPI Tx data registers. It will be a periodic circular buffer DMA channel (channel 1 and 2 provides a periodic functionality).

The SPI is configured to communicate with the external CH341 chip. The trace data received from the DMA will be transferred via this SPI bus to the external device. The SPI should be automatically called and will be configured when the MTB debug macro is configured in the main code.

The ch341 is used as the interfacing media with the host device. The received data from SPI should be transferred to host via the USB interface.

Upon reception of the trace data, the deciphering of the data is needed to make it understandable as the trace data received until here would be only Program counter values. The map file here could be used to correlate the information and a script here will map out the trace data values and actual function calls, eventually storing it to a file.

- What functionality will your project demonstrate?
  - The aim of the project is to demonstrate a bootstrap implementation of Micro trace buffer peripheral provided in the ARM Cortex M0+ architecture using the KL25Z microcontroller. Using this peripheral to trace the entire call stack history of a program execution, transmitting it via an SPI bus to an external Host device, and logging it to this device
- What technologies will you use? For those areas we have covered in class, how will you demonstrate deeper knowledge than what we covered in the biweekly homework assignments?
  - Following peripherals would be used
    1. Micro trace buffer
    2. SPI
    3. DMA
- What do you anticipate needing to learn in order to develop your project? What sources (KL25Z Reference Manual, Internet sites, etc.) do you plan to use to figure out how to do whatever it is you are attempting?
  - The aim is to learn more about the debugging architecture provided by ARM and a vision to implement a bootstrap debugger which is not limited by the use of IDE.
- Does your project require any additional hardware? If so, what will you acquire, and what is your plan for assembly? (Again, my focus is on the software you develop. I am asking about your hardware plans just so I can ensure that whatever you are planning in this area is relatively straightforward.)
  - Yes. I will be using the ch341 chip for SPI to USB implementation.
- Finally, what is your testing strategy for your project? Will you develop automated tests, will you use manual tests, or will you use a mixture of both?
  - The strategy is to implement and test each stages independently before integrating the same they would be similar to the peripherals used in the project.