Freescale Semiconductor
Application Note

Document Number: AN3253
Rev. 0, 03/2006

XGATE Library: Load Measurement

Measuring the XGATE Coprocessor Load

by: Daniel Malik
MCD Applications, East Kilbride

The XGATE Load Measurement package is a collection of header and source files, in the C programming language, that enable the user to measure loading of an XGATE coprocessor created by different application functions in real-time. The results enable the software integrator to decide whether a particular combination of application functions will satisfy the real-time requirements defined for the application.

Contents

1 Introduction

1.1 What to Measure

In principle, there are two different load related reasons why a microcontroller based system might fail to perform according to given real-time requirements.

Insufficient Performance

The majority of real-time applications are based on periodic execution of a set of tasks (e.g., periodic check of battery voltage or execution of an FIR filter). Even

freescale™ semiconductor

tasks such as the reception of a CAN message and the processing of a command it contains can be looked at as periodic — the shortest time within which the next command could be received can be used to define the maximum (period) time allowed for the processing.

Incorrect selection of software algorithms, sub-optimal high level language coding, or simply the sheer complexity of the task might, among other reasons, cause the performance of the microcontroller to be insufficient for the application.

A symptoms of this type of failure is that, on average, the system cannot perform the tasks within the specified period times (i.e., the request for the next iteration of the FIR filter comes before the last calculation has finished). Data buffers and FIFOs help to overcome problems when processing short bursts of requests; however, if the system is overloaded for a long period of time, the FIFOs and buffers will eventually overflow. This situation is highly undesirable as the system will:

- Stop processing some tasks (i.e., prioritize the tasks and drop those of lesser importance)
- Degrade the quality of service (e.g., lower the data sampling rate to reduce the number of FIR iterations required per second)
- Face a complete failure

To prevent this type of failure, the software integrator must ensure that the average load (from all the tasks combined) is below 100%. The period of time over which the average load is measured is dependent on the nature of the application itself. For example, for data logging equipment several hours may be required, whereas for a fast motor control system only a few milliseconds may be sufficient.

Long Latency Time

The majority of applications are event driven. For example, the battery voltage is measured and the FIR filter is executed in response to a timer based overflow/compare event, and the CAN message is processed in response to the event of its reception. The majority of such events are associated with the requirement of a maximum reaction time (e.g., if there is no FIFO, the FIR filter must be executed before the next data sample arrives and overwrites the current data). The software integrator must ensure that the worst case execution time of any task will not prevent the system from meeting these reaction time requirements.

Imagine a system without any data buffers where three events (*a*, *b*, and *c*) have occurred and require execution of tasks *A*, *B,* and *C*. The software integrator may decide that all three tasks have equal priorities and will be executed in the previously mentioned order. Should event *c* recur at some point in the near future, the worst case total execution time of all three tasks combined must be shorter than the shortest possible event *c* occurrence period.

Therefore, knowing only the average load that the algorithms create is insufficient for evaluation of the application performance. Knowledge of the worst case execution time of all software algorithms is also required. However, the combination of both parameters enables the software integrator to verify that the application will satisfy its design requirements.

1.2 How to Measure

1.2.1 Compile Time Code Analysis

All microcontrollers are built as complex state machines, where changes of the state are governed by a set of rules. Therefore, even for the most complex of applications, it is theoretically possible to perform a low-level simulation and calculate how the application code will perform for a given set of input events.

Microcontroller manufacturers document how individual instructions behave and how long they take to execute. In simple cases, it is possible to calculate the execution time by analyzing the assembly level output from a high level language compiler.

Unfortunately, once the system reaches a certain level of complexity, prediction of the input conditions becomes non-trivial. In systems with hundreds of events, it is close to impossible to decide what combination of events might occur at the same time and/or in which sequence. Trying to analyze all possible scenarios across all possible states of the individual application algorithms usually proves to be very time consuming and impracticable.

While analysis of the code produced by the compiler is the most accurate method of measuring the application load, it is usually only possible for applications of low complexity.

1.2.2 Real-Time Measurement

For applications where analysis of the code is impractical, the load measurement can be performed in real time. The events to which the application must react could be created artificially to represent a typical or anticipated worst case scenario. Alternatively, the performance of the application can be observed in the real-world environment for which it was created.

In such a situation a set of measurement functions is added to the application to support the analysis. Such measurement functions must provide details of the average loads and peak execution times as discussed in Section 1.1, “What to Measure”.

2 Real-Time Measurement

2.1 Measurement Algorithm

A graphical interpretation of one possible algorithm for performing the load measurement is depicted in Figure 1. The algorithm is split into two separate threads. The threads of the algorithm communicate by sharing data in memory. Both threads use a hardware timer peripheral as a timebase for the measurement.

When the application function is executed, it first calls the *lm_start* procedure of the load measurement system. This procedure records a time stamp to document when the application function started its execution.

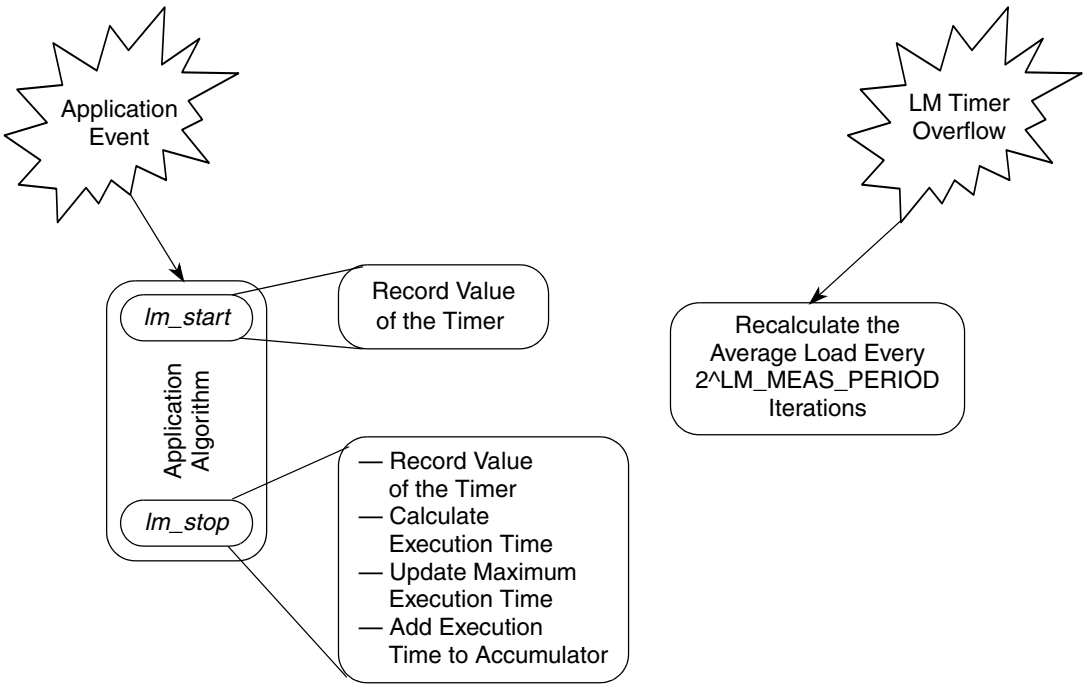

Figure 1. The Load Measurement Algorithm

Before finishing the execution, the application function calls the *lm_stop* procedure. This procedure records another time stamp and calculates the execution time of the just finished iteration of the application function. It then compares this time with the longest recorded time and updates the record in case the recent execution time proves to be the longest observed so far. The *lm_stop* procedure also adds the recently measured time to the execution time accumulator for the purpose of calculating the average load.

The average load is calculated by the second load measurement thread depicted in Figure 1. This thread is executed based on a periodical event triggered by the hardware timer peripheral. The procedure simply divides the accumulated execution time (recorded by the *lm_stop* procedure) by the period of its execution, and records the resulting average load created by the measured application function.

2.2 Implementation for the XGATE Coprocessor

The following subsections describe different aspects of a particular set of load measurement functions. These represent one possible implementation designed to aid measurement of the load created by application tasks running on the XGATE coprocessor of the S12X Family of microcontrollers. An example application, which demonstrates the use of these functions, accompanies this application note.

2.2.1 Configuration

The load measurement functions must be configured to suit the nature of the application before they are compiled and used. All configuration parameters are located in the *load_meas.h* header file.

2.2.1.1 Parameter LM_INIT_SWX

As described in Section 2.2.2, “Initialization”, the load measurement functions must be initialized before first use. Parameter LM_INIT_SWX is used to define which of the eight software generated interrupt vectors will be used to call the initialization procedure.

2.2.1.2 Parameter LM_NO_OF_CHANNELS

The load measurement is not limited to measuring performance of only one function at a time. As described in Section 2.2.3, “Measurement”, procedures *lm_start* and *lm_stop* accept a parameter that represent a particular measurement channel. The user can use different measurement channels for analyzing the performance of different application functions (or parts of a single function). The parameter LM_NO_OF_CHANNELS defines the number of channels the user wishes to use.

2.2.1.3 Parameter LM_MEAS_PERIOD

The load measurement algorithm uses a 16-bit hardware timer peripheral as an accurate time reference. The time over which the average loads are calculated is defined by the LM_MEAS_PERIOD parameter. The average calculation period is equal to (2^LM_MEAS_PERIOD)*65536 bus cycles. For example, a value of LM_MEAS_PERIOD = 10 leads to and average period of (2^10)*65536 = 67108864 bus cycles (approximately 1.678 seconds at a bus frequency of 40 MHz).

2.2.1.4 Symbols LM_USE_ECT, LM_USE_PIT, and LM_USE_MDC

The user can choose which 16-bit on-chip hardware timer will be used for the load measurement by defining one of these symbols. Therefore, it is possible to select a timer that is not used and limit the impact of the load measurement on the functionality of the application. Symbol LM_USE_ECT selects the enhanced capture timer, symbol LM_USE_MDC selects the modulus down counter (within the ECT), and symbol LM_USE_PIT selects the periodic interrupt timer.

When the periodic interrupt timer is used, additional parameters LM_USE_PIT_T and LM_USE_PIT_MT select the PIT channel and microtimer to be used by the load measurement.

2.2.1.5 Vector Table Entries

For correct functionality, the load measurement requires two XGATE interrupt vectors to be set up by the user:

1. The interrupt vector corresponding to the software trigger interrupt selected by the parameter LM_INIT_SWX (as described in Section 2.2.1.1, “Parameter LM_INIT_SWX”) must be set up to point to procedure *lm_init*. This procedure is used to initialize and calibrate the load measurement (see Section 2.2.2, “Initialization” for more details).
2. The interrupt vector corresponding to the hardware timer peripheral (selected by symbol LM_USE_ECT, LM_USE_MDC, or LM_USE_PIT — see Section 2.2.1.4, “Symbols LM_USE_ECT, LM_USE_PIT, and LM_USE_MDC”) must be pointing to the *lm_timekeeping* procedure. This procedure performs the second thread of the load measurement and calculates the average loads corresponding to the individual measurement channels.

2.2.2 Initialization

Before the load measurement procedures can be used for the first time, the load measurement algorithm must be initialized and calibrated. This is achieved by directing the load measurement interrupts to XGATE and calling the *lm_init* procedure. An example of how the load measurement can be initialized is shown in Listing 1. To simplify the initialization, the constant LM_XGATE_VECTOR and the macro LM_INIT() are pre-defined in the *load_meas.h* header file. By default, LM_XGATE_VECTOR equals zero and, therefore, the software interrupt 0 is used for the initialization.

Listing 1. Initialization of the Load Measurement (Executed on the S12X CPU)

```
/* sets interrupt priority level and routing for a given channel */
void SetIntPrio(char channel, char prio) {
     Interrupt.int_cfaddr = (channel << 1) & 0xf0;
     Interrupt.int_cfdata[channel & 0x07].byte = prio;
}

/* XGATE load measurement initialization */
SetIntPrio(LM_XGATE_VECTOR, RQST|3);        /* LM timer overflow interrupt */
SetIntPrio(0x39-LM_INIT_SWX, RQST|1);       /* LM init interrupt */
LM_INIT();                                  /* call the LM init interrupt */
```

2.2.3 Measurement

The load measurement is ready to use once the initialization is finished. For achieving accurate results the compiler must be prevented from optimizing the function calls to the *lm_start* and *lm_stop* procedures. The *load_meas.h* header file defines two macros for the purpose: LM_START(n) and LM_STOP(n). These macros implement the calls to the *lm_start* and *lm_stop* procedures in assembly language preventing the compiler from performing any optimizations. Then the function calls always take the same number of XGATE cycles to execute (independent of the compiler capabilities and surrounding application code) making the measurement more accurate. The parameter *n* should be a compile time constant and is equal to the particular measurement channel to be used. The allowed range for the measurement channel number is 0 to LM_NO_OF_CHANNELS-1. An example of the use of these macros is shown in Listing 2.

Listing 2. Example of Use of the LM_START(n) and LM_STOP(n) Macros

```
return_value XGATE_application_function(parameter_list) {
     LM_START(0);
     {
          /* declaration of local variables */
          ....
          /* body of the application function */
          ....
     }
     LM_STOP(0);
}
```

2.2.4 Reading the Results

The results of the measurements are stored into arrays *lm_load* (average loads) and *lm_peak_runtime* (maximum observed execution times). The number of elements in these arrays is equal to LM_NO_OF_CHANNELS. When reading the results out of the arrays (e.g., using a debugger), the index corresponds to the particular measurement channel.

2.2.4.1 Average Load Results

The average load results stored in the *lm_load* array are 16-bit wide unsigned integer values. The value 0 correspond to 0% load; the theoretical value 65536 corresponds to 100% load. Therefore, the percentage average load is calculated as lm_load[n]/65536*100. For example, a value of 2000 corresponds to an average load of 2000/65536*100 = 3.05%.

2.2.4.2 Maximum Execution Time Results

The maximum execution time results in the *lm_peak_runtime* array are 16-bit wide unsigned integer values. The value corresponds to the execution time expressed in bus clock cycles. For example, the value of 2000 at a bus frequency of 40 MHz corresponds to 2000/40 MHz = 50 µs.

2.2.5 Performance Requirements of the Load Measurement

The load measurement procedures require a certain time to execute and thus represent a certain amount of XGATE load themselves. While the overhead of calling the *lm_start* and *lm_stop* procedures is compensated for and subtracted from the results, the application itself must allow for the extra processing required and accommodate the associated increase in latency times. The average load, the execution time, and the required memory size of the different procedures are shown in Table 1. CodeWarrior compiler/debugger version 4.5 was used to obtain the values shown in the table. Execution times were recorded in a setup where there were no collisions between the CPU and the XGATE when accessing the on-chip RAM.

Table 1. Required Memory Size and Performance Details

Component	Execution Time or Size	Units
lm_timekeeping()	10 every 65,536 bus cycles plus 30.5n + 7 (n is the number of channels) every 2^LM_MEAS_PERIOD*65536 bus cycles	Bus cycles
LM_START(n)	9	Bus cycles
LM_STOP(n)	32 when the peak execution time is not recorded or 33 when the peak execution time is recorded	Bus cycles
Code size	304	Bytes
Data size	10n + 4 (n is the number of channels)	Bytes

3 References

MC9S12XDP512 Data Sheet, Freescale Semiconductor Inc., 2005.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3253
Rev. 0
03/2006

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor