Application Report

Embedded Real-Time Analysis and Response for Control Applications

ABSTRACT

The C2000™ family of devices focuses on real-time control. Real-time control systems can be visualized as executing some form of control loop within a stipulated amount of time. This involves sensing of one of more inputs, an algorithm to process this information and then decide on actions to be taken that are driven out of the device through some peripheral. This cycle of operation can never be delayed or stopped. With this ongoing process, it is equally important for such systems to be able to respond to any condition that can cause the system to fail or get into a state where it cannot handle the current system conditions. The ability to realize such a capability poses a complex technical challenge. This application report describes the architecture in C2000 that facilitates intelligent monitoring of CPU and System activity that provides non-intrusive diagnostic and correction capability.

The response of the system during operation is many times adaptive and specific to the nature of the system. Our architecture in C2000 facilitates intelligent monitoring of CPU and system activity that provides a completely non-intrusive diagnostic and detection capability. This is coupled with a “Configurable Logic Block”, which provides the ability to create an adaptive and user configurable response where the operation of some of the peripherals can be modified by the user.

Table of Contents

List of Figures

Trademarks

C2000™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

The critical need for any real-time closed loop control system is to remain stable throughout life, as shown in Figure 1-1, and have a way to handle any out-of-bound system scenario in the safest possible way. At the time of design, the control systems are architected to handle a range of system variables, parameters and scenarios.

When the system is in use, if any of these variables or parameters were to go beyond what the control loop is designed for, then the stability of the whole system is at stake. While real-time debug and trace techniques exist, these can only help in a development environment with a debugger or trace receiver hooked up. These technologies cannot be used once the system is actually functioning in the end equipment.

A good example of such a scenario is a motor controller in servo drive or any real time motor in an electric vehicle. In a development environment, real-time debuggers and trace technologies can be used to collect data, perform offline analysis and use this to tune and modify the system. But once this goes into a production system, the above said mechanism for on-chip monitoring is a critical requirement.

C2000 MCU architecture includes a new and advanced peripheral **E**mbedded **R**eal-time **A**nalysis & **D**iagnostic (ERAD) that facilitates sophisticated non-intrusive detection and monitoring of the MCU operation in real time. This mechanism in the architecture can run in parallel to the main CPU along with the user application. In the event of detecting a dangerous situation in the architecture, the hardware mechanism helps with a quick response to address the problem by taking the best possible corrective action runtime.

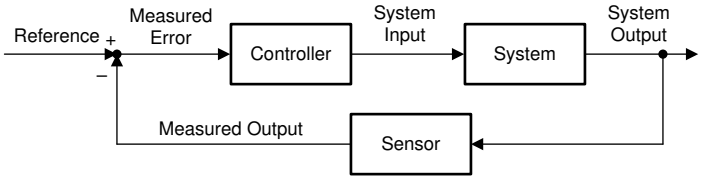

Figure 1-1. Basic Elements of a Control System

With a typical control loop, the transfer function can be described as a sequence of operations starting with the input peripherals (like the analog-to-digital converter (ADC) that creates inputs for the control routine), the critical system software running on the CPU doing the necessary computations and deciding the next set of actions on the output peripherals (like the PWM). The exact sequence and details of such system an operation will be different for different end applications. However, in all real-time systems, sequences like the one described above need to run within a certain time period. If not, there will be serious implications since they are controlling equipment's like motors and power supplies.

A lot of care is taken during the development phase to handle out of range inputs and different system use conditions. However, when these systems go into a production environment, there is certainly a possibility of encountering something beyond what the system designer had anticipated during development. While there is no expectation for the system to function properly under such out-of-bound conditions, it is quite important that this situation be handled in a graceful manner without any significant damage to the final end equipment. It is this detection and handling that poses a significant challenge in real-time systems and it is here the ERAD peripheral will come to rescue the system designer.

ERAD can be used to realize hardware breakpoints and few more profiling options are available with debugger connected, but focus is purely on run-time monitoring and diagnostics in the coming sections.

1.1 What is ERAD?

ERAD is a hardware module that can be configured to non-intrusively monitor and profile critical CPU buses and device events.

ERAD is a hardware module with enhanced bus comparators and system event counters and sits within the MCU bus architecture as shown in the overview diagram. ERAD on its own can generate system level interrupts and flags and can also feed into other peripherals such as the CLB (configurable logic block) for further enhancing the capabilities. For complete understanding of the relationship between these two modules, see the CLB documentation located in the device-specific TRM.

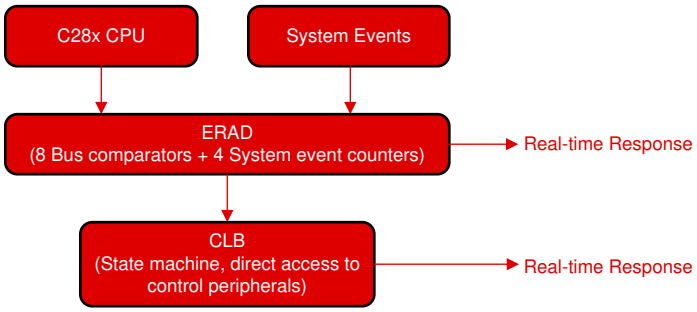

Figure 1-2. ERAD Block Diagram

1.2 Enhanced Bus Comparator

ERAD monitors the CPU interfaces in a non-intrusive manner and also monitors some critical CPU internal buses and signals that convey information about the CPU code execution and pipeline. Most of the critical CPU interfaces are monitored by a comparator unit, which can be configured to match any address/data/instruction patterns that appear on the CPU interface and generate events based on this, as shown in Figure 1-2. There are multiple such units that can generate multiple events in parallel with different pattern matching capabilities. The comparator supports bit masking while comparing, which helps to detect/trigger events on ranges and not just discrete times alone. A simple example would be to generate an event when an address of interest is accessed for a read or write.

Other examples would be to trigger an event if code access goes beyond a certain safe zone or detecting a stack overflow as shown in the examples to follow.

The following is a list of CPU interfaces that can be monitored by EBC (F28004x):

- VPC - Program Counter
- DRAB - Data Read Address Bus
- DRDB - Data Read Data Bus
- DWAB- Data Write Address Bus
- DWDB- Data Write Data Bus
- PAB - Program/Instruction fetch bus

1.3 System Event Counter

In addition to monitoring the critical CPU interfaces, monitoring various events in the system like interrupts and peripheral activity is another important requirement for any real-time system. All the key system events like interrupts, DMA triggers and important peripheral events are brought into the ERAD IP. Hence, these can be time-stamped, counted and measured with relation to the CPU activity. Using state machines in CLB, it is possible to further define sequences of operations and events. Counter block allows the system events, as well as CPU activity, to be measured and analyzed with reference to other events, too.

Simple examples of linking CPU activity and system events are:

- Completion of CPU software and PWM output generation within a specified time from an ADC data capture event
- Reception of a CAN message periodically within a maximum specified window of time

Counter supports a unique start stop mode, where in counter records the CPU cycles spent between any start and stop event. Moreover there is additional hardware which continuously keeps a record of the maximum count value reached by this counter across multiple start/stop iterations. Importantly, all the above mentioned metrics are achieved with zero software overhead.

Some of the unique benefits rendered by this counter are:

- Profiling code segments
- Temporal relationship between System events and CPU transactions
- Counting duration between specified memory accesses (Reads and Writes) and system events
- Counting of system events (like interrupts, CLA events, and so forth)
- Counting duration between system events
- Measuring the maximum time spent in between a pair of events over multiple iterations with zero software overhead, as shown in the examples to follow.

List of events that can be profiled by SEC (F28004x) are:

- PIE interrupts to CPU
- CLA Interrupt signals
- EBC generated events
- SEC generated events (SEC events from one SEC can be sent across to another)

2 Application Scenarios

2.1 Stack Overflow Detection

A frequently encountered issue with embedded code development is detecting an overflowing stack. With ERAD, a single enhanced bus-comparator unit can be used., Then, map the stack end address with a small margin to generate an interrupt when there is a write that is attempted. With the example below, just use a comparator with address 0x99FF0 and mask the last 4-bits for comparison. Any write access to 0x99FF0 to 0x99FFF would generate an event/interrupt, which can then be handled appropriately based on kind of the application.

Example: *C2000Ware_3_03_00_00\driverlib\<f28004x/f28002x/f2838x>\examples\erad \c28x_hwbp_stack_overflow\erad_ex8_hwbp_stack_threshold_detection.c*

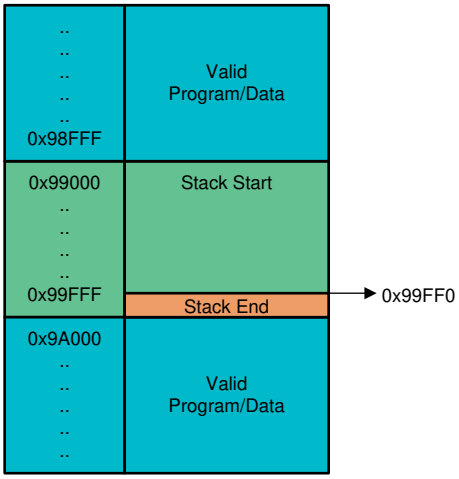

Figure 2-1. Stack Overflow Detection

2.2 Detecting Unintended Memory or Peripheral Write

The same example shown in Section 2.1 can simply be extended to a bigger region. You can certainly detect if there are any writes performed to a location or the region that is not intended to. Since these modules can be configured on the fly, you can certainly re-configure (enable/disable) these blocks at different times giving different context at which these units will be active and not.

2.3 Code Profiling During System Operation

Another interesting run-time example would be profiling sections of code during run-time without any software overhead. Additional features like threshold monitors and hardware based max counting, provides the necessary tool set for the embedded software engineer to realize more efficient control loops, critical to the overall system performance.

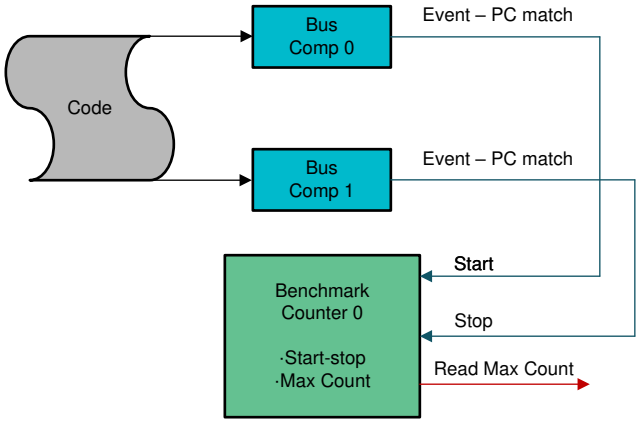

Figure 2-2. Code Profiling

Using the code diagram in Figure 2-2 as an example, this would utilize two of the eight enhanced bus comparators to generate events (to mark the start and end of that routine/function) and a counter for profiling.

Example: *C2000Ware_3_03_00_00\driverlib\<f28004x/f28002x/f2838x>\examples\erad \c28x_ctm_profile_function\erad_ex9_ctm_max_load_profile_function.c.*

Some examples of what could be done with the data collected in the example above in addition to simple profiling:

- At any given point of time, the maximum CPU cycles that were consumed by an executing code/routine is available with a single register read (MAX_COUNT).
- A threshold register can be programmed to generate an event/interrupt when the counter exceeds a pre-defined limit. (This can act precisely as a safety mechanism if you were to complete this routine in a given time period and violating this would be critical for the transfer function).

2.4 Run-Time System Monitoring, Diagnosis and Response Generation in Real-Time Systems

Thus far it has been shown that the ERAD module is acting as a monitoring agent. However, when used in conjunction with the Configurable Logic Block (CLB), it is possible to create a real-time response to various ERAD generated events. In this scenario, the ERAD defines and detects various system scenarios and the CLB drives a corrective action utilizing its connections to key peripherals such as the ePWM or direct control over a GPIO (like setting an ERROR pin).

Figure 2-3 shows a conceptual power stage control inside the C2000 MCU and how the ERAD plus CLB could be inserted in the control loop for non-intrusive monitoring as well as support with quick response to anomalies. There is potential for an out of bound condition to cause erroneous outputs to be driven out to the power stage of the system. The result of which could be a range of system effects anywhere from an inefficient system to potential damage to the power electronics.

Even if some of these anomalies cause critical interrupts to be generated, it will take a few micro-seconds for the CPU software to execute the interrupt service routine, diagnose the issue in software and generate a corrective remedial response.

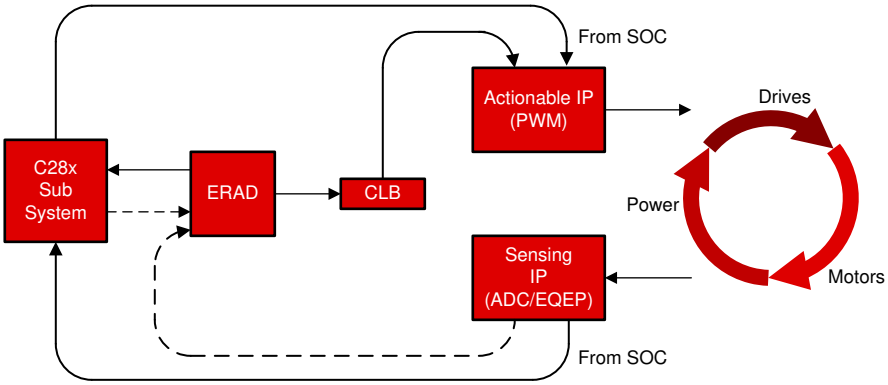

Figure 2-3. Run-Time Monitoring and Response

Furthermore, the CLB can do a second level of analysis on the generated events to create a spatial relationship profile. The temporal relationship between a specific system interrupt and time taken for the Interrupt code to be executed can be constrained with a fixed threshold and report back using another interrupt if the system load does not allow this.

For example, consider that PWM module sends an ADC conversion request via EPWMxSOCA/B and the ADC hardware senses the request and initiates a conversion followed by an interrupt to indicate the completion. This in general initiates a control loop execution based on the sensed ADC value, which needs to be executed within a prescribed time decided by the system integrator. One of the variable factors here is the interrupt being taken on time, once the ADC generates an interrupt request. There is no guarantee that the CPU will start this ISR after ADC sends the interrupt request, as it could be executing another higher priority task. ERAD/CLB combo can help define a threshold time by when the interrupt execution should have started from PWM issuing EPWMxSOC, if not create a PWM Trip.

To design a system to address the above problem using ERAD and CLB, consider that CLB gets two events:

- EPWMxSOC input comes to CLB directly, which also signals ADC to start conversion (Counter start)
- ERAD is used to create an event when the ADC ISR start address is executed (Counter stop)

Using both these events to start and stop a counter, a threshold value can be set to create a trigger event if the later/stop event does not come on time. This event can be used to generate a PWM Trip if the ADC interrupt is not taken within the prescribed time.

In summary, ERAD plus the CLB allows for critical on-chip monitoring and graceful recovery options. In most existing control systems this level of monitoring is either not present, requires a non-sustainable CPU overhead, or requires an expensive external solution. For details on its capabilities and programming, see the CLB architecture and documentation located in the device-specific TRM.

For additional details on hardware and software source that can help to jump start using ERAD, see Section 4.

3 Summary

With real-time system designs getting complex every day, it poses unique challenges for the system designer to make sure the system remains stable across the known and unknown sequence of events. The architecture of ERAD with CLB gives the system designer a unique real-time decision making and correction capability. Since both these blocks are completely user configurable, it allows different users to make their own definition of temporal and sequence relationship between their software, the interrupts and peripheral actions in the system. All of this operates independently, concurrently and completely non-intrusive to the user CPU and code. In the real-time control world, these capabilities give the C2000 devices a unique capability which does not exist in other similar platforms currently.

4 References

- Texas Instruments: *TMS320F28004x Real-Time Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F28002x Real-Time Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F2838x Microcontrollers Technical Reference Manual*
- ERAD Introduction - https://training.ti.com/embedded-real-time-analysis-diagnostics-erad-c2000-devices
- ERAD C2000ware examples -
  - C2000Ware…/driverlib/f28004x/examples/erad
  - C2000Ware…/driverlib/f28002x/examples/erad
  - C2000Ware…/driverlib/f2838x/examples/c28x/erad
- Texas Instruments: *Designing With the C2000™ Configurable Logic Block (CLB)*
- CLB Tool intro - https://training.ti.com/c2000-configurable-logic-block-clb-programming-tool

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265