F2837x Driverlib to F2838x Driverlib - Migration Guide

Contents

Introduction

This document provides information on migrating driverlib from F2837x device to F2838x device. C2000WARE v2.0 adds support for F2838x series of devices.

COFF to EABI Transition

In addition to support for F2838x series of devices, one of the other main updates incorporated in C2000WARE v2.0 is the transition from COFF to EABI as the binary interface standard for programs and program components. There might be certain changes required to the existing COFF ABI application or library source code to be compatible with EABI. The article provided in the link below covers some of the key aspects to be considered while migrating to EABI.

C2000 EABI Migration

In addition, the section below this document on Driverlib migration notes describe few of the specific updates needed to be done to driverlib and some of the C2000 software library source code as part of transition to EABI.

Index Library Approach

Indexing approach is now being used across many libraries including driverlib, FPU DSP, FPU FASTRTS, IQMath, SFO etc. This is being done to enable abstracting out library info into an library index and then linking against index instead of linking directly to individual coff or eabi libraries, thus helps in efficient management of many versions of the same library. For example if there exist multiple versions of the same object file libraries, each built with different set of build options then the library information archiver can be used to create an index library of all the object file library versions and this index library is used in the linker in the place of particular version of object library. The linker looks at the build options of the application being linked and automatically decides which specific version of library output file need to be used. Please refer to TMS320C28x Optimizing C/C++ Compiler User's Guide http://www.ti.com/lit/SPRU514 for more details on indexing approach. The same naming convention is followed across all the libraries while applying indexing, for example **driverlib_coff.lib** and **driverlib_eabi.lib** are the COFF and EABI variants while **driverlib.lib** is the index library which needs to be used in project's linker options. Indexing approach will help the customers to transition from COFF to EABI very smoothly and play around with it conveniently without breaking the backward compatibility as the project linker options need not to be updated while switching between COFF and EABI.

NOTE : Do take care while adding/copying the libraries (where indexing is used) to the CCS projects i.e. all the versions (COFF, EABI) of the library including the index library need to copied to the projects so as to allow the linker to pick the correct version of the library. This step is not required if the libraries are being linked through project build options. Thus the recommended practice is to always link the libraries instead of copying. Also please refer to individual user guides for the libraries while linking their index.

Driverlib Migration

Along with the driverlib for C28x core and peripherals, F2838x has a driverlib for the CM subsystem peripherals. This CM driverlib is a new component, but maintains similarity to C28x driverlib in terms of API definition styles and code organization.

For C28x driverlib, there are few changes from F2837x to F2838x and the following section highlights the main differences.

- **System Peripherals**
  - SYSCTL
    - New Features added in IP :-
      - All the APIs with SYNC_SOC registers pertaining to SYNCIN have been removed due to change in the syncin flow
      - New set of NMI flags have been added
      - New reset causes added
      - New ePWM SOC signals added for driving an external ADCSOC signal.
      - New Peripheral clocking added
      - New Peripheral Resets added.
      - New Peripherals for CPU selection
      - New peripheral access controls added
      - New Clock out sources added.
      - New Sync out sources added.
      - The System and Aux PLL have new configuration parameters to lock the PLL.
      - The sequence of PLL locking has been modified.
      - The PLL Validation now has 1 more parameter of pllclk to decide which PLL is to be validated SYSPLL or AUX PLL.
      - Added Single ended XTAL as clock source for AUXPLL.
      - AUXPLL clock divider now has 4 more values.
      - Removed need to configure whether whether the dual ported bridge is connected with DMA or CLA as the secondary master.
    - Enhancements to pre-existing APIs :-
      - The config parameter provided for setting clock have been updated.
      - Updated the pollX1cnt API to make the clearing of the counter to be asynchronous.
      - The API to get system clock has been optimized to calculate clock from the new multipliers , remove the switch case and use a simple mask and shifting operations.
      - Adding checks for multiplier values to prevent unnecessary locking of PLL during PLL setup.
      - Update the API to get Aux clock to use the correct clock source select register. Added custom code to calculate Integer multiplier as divide values are randomized . Clock source is only checked with OSC2 and not OSC1 now.
      - The APIs to turn on / off / select oscillators now uses both the Clock Source Control as well as the XTAL control registers to turn on/off the XTAL .

        - The clock verification on setting up clock for both System clock and Aux Clock is done using a DCC rather than the CPUtimer used earlier.
        - The Boot ROM Status Address and Reset Status flag defines have been updated . Now it is same for both CPU1 and CPU2.
        - Peripheral clocking name for HRPWM changed to HRCAL .
    - Addition of new APIs :-
        - To control and configure CPU2/ CM reset
        - CM to CPU NMIs/Interrupts
        - CPU1 error interrupts & interrupt masks
        - Request to stop peripheral clocks
        - Simulate reset
        - Select the BANK to be programmed by CPU1
        - Allocate shared peripherals
        - Setup clocks to newly added peripherals like CMClk , EnetClk , EcatClks , MCANClks etc.
        - Lock config registers
        - Peripheral access control and other system control functionalities .

- DCC
    - New Features added in IP :-
        - Driver was first added in F28004x
        - Addition of new clock sources for counter 1 and counter 0.
    - Enhancements in pre-existing APIs :-
        - DCC clock source 1 now of 5 bits rather than the 4 bits in F28004x.
        - Implementation of APIs used to get counter1 and counter0 value have been updated to read all 32 bits in one shot to prevent read of incorrect data.
- GPIO
    - New Features added in IP:-
        - To read data from GPIO Data register. Refer the new APIs GPIO_readPinDataRegister() and GPIO_readPortDataRegister() in API Guide for more details.
        - A new enum value GPIO_CORE_CM is added in GPIO_CoreSelect to select CM as the master core.
- Interrupt
    - No change in the IP.
    - Addition of new APIs in the driver
        - Separate APIs have been added to enable/disable PIE module. Earlier this was done only as part of init API. Since the init API disables all the interrupts as well as part of init sequence, this API has been added to support enabling/disabling of PIE module individually if required by an application.
- CPUTimer
    - No change in the IP
- XBAR
    - New Features added in IP :-
        - Added a new X-Bar Input Flag Register 4.
        - No. of X-BAR input being configured has increased to 16.

- Addition of new mux signals for the input , output & epwm xbars.
- Enhancements to pre-existing APIs :-
  - Addition of a new parameter " base " for all the output & input xbar APIs.

- **Security**
  - DCSM
    - New Features added in IP :-
      - Addition of new RAMs (RAML6 and RAML7) whose execute only status can be read.
      - Addition of Blocked mode as a Zone control status
    - Enhancements to pre-existing APIs :-
      - Flash sector values have become Sector 0 to Sec 13 from sector A to N .
      - Addition of a parameter for CPU selection in get Zone and get Status APIs which had separate register spaces for different cores previously.
      - Implementation of get Zone Control Status APIs updated have register accesses of all Zone control registers which have become 32 bit compared to 16 bit.
      - Register name changed from EXEONLYRAMR to EXEONLYRAM1R
      - ZSB calculation of link pointers updated to use 14 bit Link Pointer and not 29 bit Link pointer as in prior devices.
    - Addition of new APIs :-
      - Dummy read of CSM password of a zone
      - Read OTP secure lock status of a zone.
      - Write Zone CSM key.

- **Hardware Accelerators**
  - DMA
    - No changes in the IP
    - Updated the trigger sources as per the device
  - CLA
    - No changes in the IP (compared to F28004x device)
    - Updated the trigger sources as per the device

- **Memory**
  - BGCRC
    - New driver
  - EMIF
    - No change in IP(from F2837x)
    - **SDRAM mapped addresses are mapped to 22-bit addressing range -** In this device a portion of EMIF1-SDRAM (0x80000000 - 0x800FFFFF) will be dual mapped to 0x00200000-0x002FFFFF (2MB) which is in the 22 bit address range to generate optimal code. Refer driverlib EMIF example for accessing SDRAM through address in 22-bit addressing range.

- FLASH
  - **New APIs added to clear the position of single bit error:**
    1. Flash_clearLowErrorPosition: Clears the error position bit of the lower 64-bits for a single bit error.
    2. Flash_clearHighErrorPosition: Clears the error position bit of the higher 64-bits for a single bit error.

- Memcfg
  - New Features added in IP
    - **Support for new message RAMs-** Additional message RAMs have been added in this device for communication between different cores & hardware accelerators. Hence existing driver APIs have been modified & new macros/enums have been added to support these additional RAMs. Refer device TRM & API guide for more details.
    - **Addition of ECC capability in LSx RAMS-** All the LSx RAMs in this device will be ECC memories in contrast to parity memory in prior devices. This has been done to allow users to run CLA code from ECC memory. In addition, CLA1 debug accesses to LSx memories have also been allowed in this device. Support for this has been added in the driver as well. Refer device TRM for more details.
    - **Addition of LS6 & LS7 memories-** Support for these additional RAMs has been added in the driver APIs. Refer API guide for more details.
    - **Lock protection to RAM test registers-** Lock mechanism have been added for RAMTEST registers in this device and hence new APIs have been added in the driver to support this feature. Refer API guide for more details.
    - **Addition of fetch and write protection bits to M0, M1 memories and CPU1 to CPU2 message RAMs-** Support for this feature has been added in driver by modifying the existing APIs. Refer API guide for more details.
    - **Addition of DMA read as non-master violation source-** Existing APIs related to violation interrupt configuration have been modified & new macro has been added to support this feature. Refer API guide for more details.
    - **Addition of Diagnostic mode for RAM memories for testing the ECC/parity logic-** Support for this feature has been added in the driver by modifying existing APIs to support the additional test mode and adding a new API for handling diagnostic errors. Refer API guide for more details.
    - **Configuration for ROM & peripheral memories-** Support for this feature has been added in driver by adding new APIs, modifying the existing APIs & adding macros related to ROM & peripheral memories. Refer device TRM & API guide for more details.
    - **Addition of parity bits to ROM to address diagnostic coverage and testing of parity logic-** Support for this feature has been added in driver by adding a new API to force error in ROM memory & modifying the existing APIs. Refer device TRM & API guide for more details.

        - **Addition of EtherCAT memory read error as uncorrectable error-** Support for ECAT memory read as additional uncorrectable error source has been added in the driver by adding a new macro & modifying the existing APIs.
    - Porting considerations
        - Macro MEMCFG_SECT_MSGCPUTOCPU has been changed to MEMCFG_SECT_MSGCPUTOCPU0 in driver to support multiple instances of CPU to CPU RAM.

- **Analog Peripherals**
    - ADC
        - No change in IP
        - Driver is modified to support additional trigger sources for new EPWM instances added in this device.
    - ASYSCTL
        - No Change in the IP.
    - CMPSS
        - No change in the IP.
    - DAC
        - No change in the IP.

- **Control Peripherals**
    - eCAP
        - New Features added for this device
            - **Sync-in Source Select** : A Sync Select Register has been added for each eCAP to select external Sync Select. Every eCAP can have separate Sync Input Select.
            - **Input Trigger Sources** : ECAP input trigger source are added.
            - **Added Event Reset Filter** : This will clear the event filter such that no remnants are present and will also clear modulo counters any pending interrupt flags.
            - **Modulo Counter Status** : A bit has been added for the Modulo counter status.
            - **DMA interrupt :** Seperate DMA Interrupts have been added to select one of the four capture interrupts.
            - **EALLOW protection to critical registers :** EALLOW protection has been added for critical registers.
        - Addition of new APIs :-
            - To set up the source for sync-in pulse.
            - To select ECAP Input.
            - To reset the counters.
            - To set the DMA Source
            - To get the modulo counter status.

- HRCAP
    - IP same as F28004X.
    - **High Resolution capture (HRCAP)** feature has been combined with the ECAP module. This feature is enabled only on 2 instances of ECAP(Instance Numbers 6 and 7).
    - Addition of new APIs :-
        - API to return calibration status.
        - API to force calibration flags.

- ePWM
    - New features added for this device
        - **Addition of more instances**- Additional EPWM instances have been added in the device. Support for the same is added in the driver.
        - **EPWM to work at max SYSCLK frequency**- In F2837X, maximum EPWM frequency was limited to 100MHz. In this device, it has been increased to 210MHz. Support for the same is part of sysctl driver. Refer device TRM & API guide for more details.
        - **Independent PWM action configuration on CBC and OST trip event**- This feature is now being supported by adding a latch for DC events to enable CBC like signal generation on DCEVTA/B.force signals. And since independent actions can be configured for trip events and DC event this can be achieved. New APIs have been added to set the DC CBC latch mode, clear event and get the latch status. Refer device TRM & API guide for more details.
        - **Simplified Syncing scheme**- This device supports a generic syncing scheme in contrast to daisy chain based SYNC mechanism in prior device, in which any EPWM/ECAP can be master SYNC source to a sub-set of EPWM/ECAPs. In this scheme, SYNCI to SYNCO path will be broken. Syncin sources can be selected individually & desired syncout sources can be enabled. Also DCA/BEVTs has been added as syncout sources. APIs have been added to select the desired syncin & syncout sources in the driver.
        - **One shot syncout synchronized to one-shot global load of shadow registers**- In this device one shot syncout signal can be triggered from both oneshot sync & reload events. An API has been added to configure the one-shot syncout trigger. Refer device TRM & API guide for more details.
    - Porting Considerations
        - EPWM_setSyncOutPulseMode() API is not being supported in the driver. Use EPWM_enableSyncOutPulseSource()/EPWM_disableSyncOutPulseSource() to enable/disable any of the desired syncout sources.

- HRPWM
    - No change in IP(from F2837X)

- eQEP

  - New features added in hardware :-
    1. **Support for more QEP signaling modes** :- A new block has been added i.e. Quadrature Mode Adapter **(**QMA) which allows to interface encoders other than quadrature type as well. This module modifies the QEPA and QEPB signals (which don't resemble the traditional quadrature phase shifted signals) such that the modified signals resemble "directional count mode" signals, and the directional count mode of eQEP can be re-used. Thus new APIs and enums are added to enable/ disable and configure this module. Also QMA error handling APIs are also included in the driver. Refer to API guide for more details.
    2. **Latching Position count on ADCSOC from PWM module :-** ADCSOC can also trigger the latch of position counter value. Refer API guide for the new APIs which are added to configure this.
    3. **eQEP Source Selection :-** Other than just the device pins, each of the eqep input signal source now can be configured as any of the specified 16 sources. These sources include various instances of PWMXBAR and CMPSS. This allows to use the internal signals like PWM output as the source for eQEP inputs. New APIs and enums are provided to choose the desired source. Refer to API user guide for more detailed information.
    4. **Direction enhancement during index :-** This configuration can be enabled using the API described in the user guide if the end application allows change of direction during index output. By default it is disabled.

  - Enhancements in pre-existing APIs :-
    1. Earlier single API was available to enable eQEP unit timer and for loading the period value. Now an additional API is provided which can be used just to load the period value of unit timer without enabling it every time. Refer to API user guide for details about this API.

- SDFM

  - Type-1 SDFM with improvements has been added in this device
  - Type configurability- Either type 0 (F2837X) or type 1(F28004x) SDFM can be configured on this device. This has been done to retain compatibility from prior devices and should be decided once for an application. This configuration is part of sysctl driver. Refer device TRM & API guide for more details.
  - Key differences between type0 (F2837X) and type1 (F28004X)
    - FIFO has been added in type 1 and 4 new interrupts (DRINT or INT_SDFMxDRy) have been added which can be triggered on FIFO level or when filter is ready with data. Other interrupt sources like threshold events & clock failure are tied to SDINT interrupt (INT_SDFM1) in type 1.

      - In type 0, FIFO is not present and there is single interrupt (SDINT) which is tied to all the sources like data ready condition, threshold events & clock failure.
    - New features added in type1 SDFM(from F28004x)
      - **Additional PWM sync sources-** Support for sync trigger from additional PWM instances added in this device has been added in the driver.
      - **Clock multiple filters from filter1 clock-** This has been done to save the number of pins used for SDFM. New APIs have been added to support this feature in the driver. Refer API guide for more details.
      - **Enhancements to alleviate noise sensitivity of SDFM-** This had been done by adding an optional synchronizer on clock & data lines and by adding a filter on comparator events. New APIs have been added to support these features in the driver. Refer device TRM & API guide for details.
      - **Addition of two more comparators to filters-** This has been done to enable configuration of 2 threshold levels both high & low which finds application in scenarios like over-current protection. New APIs have been added & existing APIs have been modified to configure threshold 2 & event filter. Refer device TRM & API guide for more details.
      - **Enhancements to Manchester Mode Decoding-** In this device, enhancements have been done to enhance the valid frequency range of Manchester data stream. Support for this feature has been added in driver by adding new API to select clock source. Refer device TRM & API guide for more details.
    - Porting Considerations
      - highThreshold/ lowThreshold parameter in SDFM_setCompFilterHighThreshold / SDFM_setCompFilterLowThreshold has been changed from uint16_t to uint32_t to accept both threshold 1 & 2 values in single parameter. The upper 16-bits represent the threshold 2 value while lower 16-bits represent the threshold 1 values. Refer API guide for more details.

- **Communication Peripherals**
  - CAN
    - No Change in the IP.
  - EtherCAT
    - New IP and new driver
  - FSI
    - IP same as F28004X.
    - New features added:
      - **Tag-Match Scheme** - This notification scheme has been added to allow users to setup a notification whenever frames with certain TAGs are received. This feature can be used with or without the multi-slave configuration. New APIs have been added to support this feature. Refer device TRM, API guide & examples for more details.
      - **Multi-Slave configuration**- This configuration has been added to support scenarios where multiple slaves are controlled by a single master. In order to

use the FSI module in a multi-slave configuration, the slave device must utilize both tag matching, and the CLB module. To support this feature new APIs have been added to configure multiple-slave configuration. Refer device TRM & API guide for more details.

- I2C
    - Addition of new API :-
        - Support I2C Extended Mode.
- McBSP
    - No change in the IP.
- SCI
    - No change in the IP
- SPI
    - No changes in the IP
    - Increased number of SPI instances
- IPC
    - There are no new features added in the IP. There are 3 instances of IPC modules in this device - one for each pair of cores. Based on the cores, the register names and offsets are different compared to F2837X device
    - There was no driverlib available for F2837x device in C2000ware. This migration guide compares the APIs with the F2837X bit fields functions.
        - F2837X bit field functions provides support for the following commands:
            - Data Read (protected and unprotected)
            - Set Bits (protected and unprotected)
            - Clear bits (protected and unprotected)
            - Data write (protected and unprotected)
            - Function call
            - Request Memory access
            - Data Block read (protected and unprotected) (Main driver only)
            - Data Block write(protected and unprotected) (Main driver only)
            - Send generic message
        - The above functions are provided with and without the usage of message queues
        - F2838x driverlib API does not support any predefined commands. It provides API to send and receive generic commands with and without the usage of message queues.
        - Message queue handling is ported from F2838x bit field functions
        - Since there are multiple instances of IPC, all the functions expect a parameter indicating the IPC type that defines the local and remote cores
        - API mapping table:

- **Driver_Lite :**

F2837x Driver API	F2838x Driverlib API
IPCLiteLtoRGetResult	IPC_readResponse
IPCLiteLtoRDataRead IPCLiteLtoRSetBits IPCLiteLtoRSetBits_Protected IPCLiteLtoRClearBits IPCLiteLtoRClearBits_Protected IPCLiteLtoRDataWrite IPCLiteLtoRDataWrite_Protected IPCLiteLtoRFunctionCall IPCLiteReqMemAccess	IPC_sendCommand (single function to send a generic command)
IPCLiteRtoLDataRead IPCLiteRtoLSetBits IPCLiteRtoLSetBits_Protected IPCLiteRtoLClearBits IPCLiteRtoLClearBits_Protected IPCLiteRtoLDataWrite IPCLiteRtoLDataWrite_Protected IPCLiteRtoLFunctionCall	None (Since driver does not support sample commands) Provides APIs to read the command (IPC_readCommand) and send response (IPC_sendResponse)
None	IPC_readCommand IPC_sendResponse

Main Driver :

F2837x Driver API	F2838x Driverlib API
IPCInitialize	IPC_initMessageQueue
IpcPut	IPC_pushMessageToQueue
IpcGet	IPC_readMessageFromQueue

F2837x Driver API	F2838x Driverlib API
IPCLtoRDataRead IPCLtoRDataRead_Protected IPCLtoRSetBits IPCLtoRSetBits_Protected IPCLtoRClearBits IPCLtoRClearBits_Protected IPCLtoRDataWrite IPCLtoRDataWrite_Protected IPCLtoRBlockRead IPCLtoRBlockWrite IPCLtoRBlockWrite_Protected IPCLtoRFunctionCall IPCLtoRSendMessage IPCReqMemAccess	None (Since driver does not support sample commands) Provides and API to send a generic command using message queue (IPC_pushMessageToQueue)
IPCRtoLDataWrite IPCRtoLDataWrite_Protected IPCRtoLDataRead IPCRtoLDataRead_Protected IPCRtoLSetBits IPCRtoLSetBits_Protected IPCRtoLClearBits IPCRtoLClearBits_Protected IPCRtoLBlockRead IPCRtoLBlockWrite IPCRtoLBlockWrite_Protected IPCRtoLFunctionCall	None (Since driver does not support sample commands) Provides and API to read a command from the message queue (IPC_readMessageFromQueue)

Driver_Util :

F2837x Driver API	F2838x Driverlib API
IPCRtoLFlagAcknowledge	IPC_ackFlagRtoL
IPCRtoLFlagBusy	IPC_isFlagBusyRtoL
IPCLtoRFlagBusy	IPC_isFlagBusyLtoR
IPCLtoRFlagSet	IPC_setFlagLtoR
IPCLtoRFlagClear	IPC_clearFlagLtoR
IPCGetBootStatus	IPC_getBootStatus

F2837x Driver API	F2838x Driverlib API
IPCBootCPU2	None (provides a generic function to set the boot mode)
Other utility functions	IPC_waitForFlag IPC_waitForAck IPC_sync IPC_setBootmode IPC_getBootmode IPC_setBootStatus IPC_getCounter

- **Others**

Notes From Migrating Driverlib To EABI

Update floats and doubles in hw_types.h to match EABI definition

```
//*****************************************************************************
// typedefs
//*****************************************************************************

#ifndef C2000_IEEE754_TYPES
#define C2000_IEEE754_TYPES
#ifdef __TI_EABI__
typedef float         float32_t;
typedef double        float64_t;
#else // TI COFF
typedef float         float32_t;
typedef long double   float64_t;
#endif // __TI_EABI__
#endif // C2000_IEEE754_TYPES
```

Updates to Assembly Code

Removing the COFF underscore as described here
Define SysCtl_delay in terms of EABI

```
//
// Define to isolate inline assembly
//
#define SYSCTL_DELAY        __asm(" .if __TI_EABI__\n"\
                                  " .asg    SysCtl_delay     , _SysCtl_delay\n"\
                                  " .endif\n"\
                                  " .def _SysCtl_delay\n"                       \
                                  " .sect \".TI.ramfunc\"\n"                    \
                                  " .global  _SysCtl_delay\n"                   \
                                  "_SysCtl_delay:\n"                            \
                                  " SUB    ACC,#1\n"                            \
                                  " BF     _SysCtl_delay,GEQ\n"                 \
                                  " LRETR\n")
```

EABI passes structures differently. If you have pass a structure by value to a function, COFF will pass it by reference through XAR6 or the stack while EABI will actually pass it by value

Updates to Linker command files

Updating section specifications to match EABI

```
#if defined(__TI_EABI__)
   .init_array     : > FLASHC,     PAGE = 0
   .const           : > FLASHB,     PAGE = 1
#else
   .pinit           : > FLASHC,     PAGE = 0
   .econst          : > FLASHB,     PAGE = 1
 #endif //__TI_EABI__


#if defined(__TI_EABI__)
   .bss              : > RAMGS23,    PAGE = 1
   .bss:.cio         : > RAMLS4,     PAGE = 1
#else
   .cio              : > RAMLS4,     PAGE = 1
   .ebss             : > RAMGS23,    PAGE = 1
   .esysmem          : > RAMLS7,     PAGE = 1
#endif //__TI_EABI__
```

Linker and Uninitialized Section
With EABI, the compiler will initialize uninitialized sections in C. This is to support the C language which states that uninitialized global variables are initialized to zero.
This can cause a couple issues.

Header File Structures

The first and most important is when using peripheral/register structs in header files and the linker command files associated with those header files. All of the structs are uninitialized, so when compiling with EABI,

those registers will get initialized to zero during the *__c_int00* code. This causes the CPU to write to those registers. When writing to DCSM registers (DCSM key registsers), this has caused a reset on the device, and the __c_int00 code was unable to compelete and get to main().
Therefore, certain sections and perhaps all register sections from the header file structures should be NOINIT. In the linker command file you can add "type = NOINIT" the memory sections you wish to not have initialized. For example:

```
  DcsmZ1RegsFile        : > DCSM_Z1,        PAGE = 1, type=NOINIT
```

Message RAMs

The second potential issue is concerning message RAMs. The CPU is not able to write to some message RAMs which are only meant to be read by that CPU. In this case, we should add NOINIT to the message RAMs since writing to memories which are read only is a waste of time. Additionally, there are RAM init bits available for the application to use to initialize the data in the message RAMs

Keeping Unused Sections

Opposite of COFF, EABI/ELF's default behavior of the linker is to remove all un-referenced sections. There isn't an option to revert the behavior. There is a linker option, **--unused_section_elimination=off**, that will keep all un-referenced sections in a link. This is close, but doesn't quite match up with COFF because COFF supported a .clink directive that marked secitons as conditionally linked. The **--unused_section_elimination** option would keep those sections as well which would result in code size growth. The compiler will put functions and data into .clink sections.

Your best path forward is to use .retain and RETAIN for un-referenced sections. There is also a linker option, **--retain**, as well.

For assembly use: .retain "<name of section>"

For C use: Use #pragma RETAIN("<name of section>")

In Linker: Use **--retain "<symbol name>"** or **--retain '(section name)'**

controlSUITE to C2000Ware Transition

C2000Ware for C2000 microcontrollers is a cohesive set of development software and documentation designed to minimize software development time. Refer below link for more details while migrating from controlSUITE to C2000Ware.

controlSUITE to C2000Ware Transition Guide