**Synchronous Cyclic Redundancy Check**

**FDD #CM-800A**

[1. High Level Description 3](#_Toc440433823)

[2. Derived Requirements 3](#_Toc440433824)

[3. Sub-Function Data Flow 3](#_Toc440433825)

[4. Design Rationale 3](#_Toc440433826)

[5. Sub-Functions 4](#_Toc440433827)

[5.1. Management of CRC Hardware Units 4](#_Toc440433828)

[5.1.1. Initialization of Management RAM (SyncCrcInit0) 5](#_Toc440433829)

[5.1.2. RTE Initialization (SyncCrcInit1) 5](#_Toc440433830)

[5.1.3. Sub-Function: RelsCrcHwUnit 5](#_Toc440433831)

[5.1.4. Sub-Function: GetAvlCrcHwUnit 6](#_Toc440433832)

[5.1.5. Sub-Function: ResvCrcHwUnit 7](#_Toc440433833)

[5.2. SyncCrc API Functions 9](#_Toc440433834)

[5.2.1. Sub-Function: 32-Bit Ethernet CRC 10](#_Toc440433835)

[5.2.1.1. Hardware Related Design 10](#_Toc440433836)

[5.2.1.2. Software Related Design 10](#_Toc440433837)

[5.2.1.2.1. Calc32BitCrc\_u08 10](#_Toc440433838)

[5.2.1.2.2. Calc32BitCrc\_u16 11](#_Toc440433839)

[5.2.1.2.3. Calc32BitCrc\_u32 11](#_Toc440433840)

[5.2.2. Sub-Function: 16-Bit CRC 12](#_Toc440433841)

[5.2.2.1. Hardware Related Design 12](#_Toc440433842)

[5.2.2.2. Software Related Design 12](#_Toc440433843)

[5.2.2.2.1. Calc16BitCrc\_u08 12](#_Toc440433844)

[5.2.2.2.2. Calc16BitCrc\_u16 12](#_Toc440433845)

[5.2.3. Sub-Function: 8-Bit SAE-J1850 CRC 13](#_Toc440433846)

[5.2.3.1. Hardware Related Design 13](#_Toc440433847)

[5.2.3.2. Software Related Design 13](#_Toc440433848)

[5.2.3.2.1. Calc8BitCrc 13](#_Toc440433849)

[5.2.4. Sub-Function: 8-Bit 0x2F CRC 14](#_Toc440433850)

[5.2.4.1. Hardware Related Design 14](#_Toc440433851)

[5.2.4.2. Software Related Design 14](#_Toc440433852)

[5.2.4.2.1. Calc8BitCrc0X2F 14](#_Toc440433853)

[5.3. Sub-Function: AUTOSAR API Wrapper 15](#_Toc440433854)

[5.3.1. Sub-Function: Crc\_CalculateCRC32 15](#_Toc440433855)

[5.3.2. Sub-Function: Crc\_CalculateCRC16 15](#_Toc440433856)

[5.3.3. Sub-Function: Crc\_CalculateCRC8 15](#_Toc440433857)

[5.3.4. Sub-Function: Crc\_CalculateCRC8H2F 15](#_Toc440433858)

[6. Timing / Execution Constraints 16](#_Toc440433859)

[6.1. Rationale / Comments 16](#_Toc440433860)

[6.2. Rates and State Execution 16](#_Toc440433861)

[7. Serial Communications Interfaces 16](#_Toc440433862)

[8. Additional Information 16](#_Toc440433863)

[9. Revision Record & Change Approval 17](#_Toc440433864)

# High Level Description

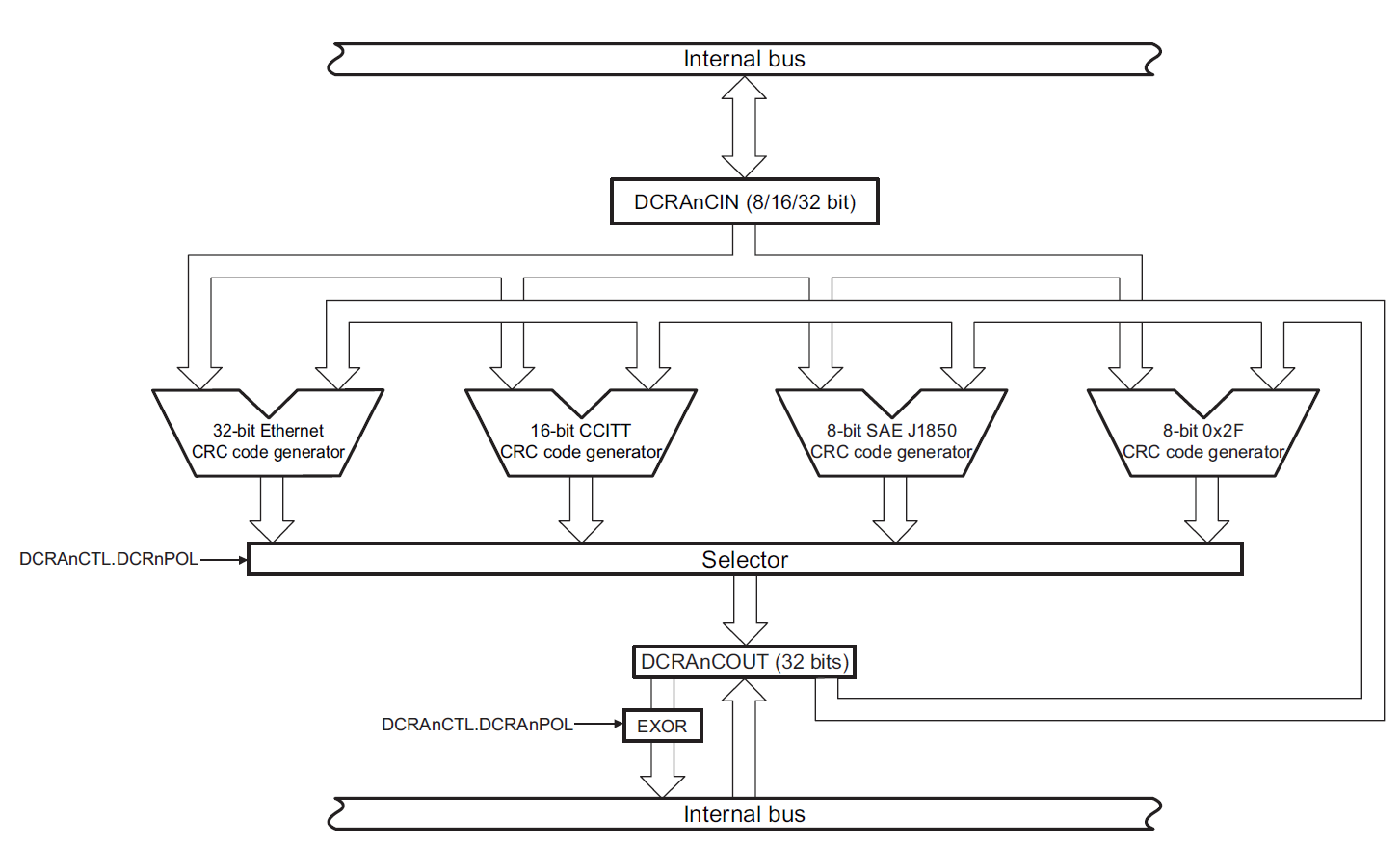
This document describes the design of the application programming interface (API) to allow application software components (SWCs) to interact with the cyclic redundancy check (CRC) hardware peripheral included in the microcontroller in EA4 hardware.

# Derived Requirements

N/A

# Sub-Function Data Flow

The follow block diagram depicts how the data flow inside the microcontroller CRC peripheral.



# Design Rationale

The design of the API was intended to meet the AUTOSAR CRC API definition as closely as possible. This allows the software configuration to utilize the hardware instead of using software libraries to save on throughput for software calculations. While the direct API does not match the AUTOSAR API definitions, wrapper functions are included to interface with components using the AUTOSAR API with the SyncCRC API.

# Sub-Functions

## Management of CRC Hardware Units

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_48, CM800A\_49, CM800A\_50, CM800A\_67, CM800A\_68, CM800A\_69, CM800A\_70, CM800A\_81***

The software implementation shall provide a mechanism to utilize one of the CRC hardware units for a synchronous CRC calculation from a software component calling one of the API functions. Upon completion of the job, the software shall release the hardware unit to be available by another software component. This management shall be implemented by a RAM table that has holds the task ID and the of the CRC hardware index. No periodic function is required to manage the RAM since the calculations are synchronous, released at the end of the API call, or permanently reserved. Furthermore, each API function will update the RAM after the job has been completed.

The implementation shall also provide a way to reserve one or more units to dedicate to a particular function if a program requires. The reservation shall be done by the pre-compile configuration or by a function call. The pre-compile configuration shall provide a permanent reservation of the CRC hardware unit. The reservation shall start from the highest hardware index. Any hardware CRC unit that is permanently reserved shall not be used by the SyncCRC API or a temporary reservation of a hardware CRC unit and should be considered as not enabled or available.

The function call shall provide a temporary allocation of one of the available, non-permanently reserved, CRC hardware units. This CRC hardware unit shall be reserved until the calling function releases it. The details of this function are described later in this document.

The management shall also provide protection from preemption of higher priority tasks by utilizing the OS task ID of the calling function as an authority to use that CRC hardware unit.

An example of the RAM table is shown below. Hardware index 0 is temporarily reserved. Hardware index 1 is assigned to task ID 2 until the calculation has completed. Index 2 is available for the next caller. Hardware index 3 is permanently reserved and not available to software applications invoking the API, but is be available to a dedicated source if required by the program.



### Initialization of Management RAM (SyncCrcInit0)

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_71, CM800A\_50***

The RAM shall be initialized according to the following pseudo code. This allows the pre-compile configuration to block access to the CRC hardware units that are not available to the application software components. In order for this to be effective, the initialization is required to be scheduled before any software components invoke the API. This function shall be called from outside of the RTE during “cold init.”

For each CRC Hardware Unit:

CrcHwSts[HwUnit].TaskId = Invalid Task ID

If HwIdx < Number of Active Hardware Units:

CrcHwSts[HwUnit].CrcHwSts = Available

Else:

CrcHwSts[HwUnit].CrcHwSts = Crc Not Enabled

End If

End For Loop

### RTE Initialization (SyncCrcInit1)

This function stub is required to properly place the SyncCrc component within the correct application within a program. This function is called by the RTE during initialization.

### Sub-Function: RelsCrcHwUnit

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_48***

This sub-function shall be used by the API to release a CRC hardware unit. This shall be called after the CRC calculation is complete for any of the API functions calls. The operation of the function shall meet the following pseudo code.

Function Inputs:

CrcHwIdx := This value represents which hardware index the action should go against.

Function Outputs:

Void

Function:

CrcHwSts[CrcHwIdx].TaskId = Invalid Task ID

CrcHwSts[CrcHwIdx].CrcHwSts = Available

### Sub-Function: GetAvlCrcHwUnit

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_48, CM800A\_67, CM800A\_81***

This sub-function shall be used by the API to allocate a CRC hardware unit for the caller. This shall be called at the start of any of the CRC API functions calls. The operation of the function shall meet the following pseudo code. The ReserveTaskId is a set of four (4) known values that will be used for the task ID of a hardware unit that is temporarily reserved.

Function Inputs:

ResvCrcCall := Input to decide if the call is for a Crc unit reservation or a standard synchronous

calculation.

Function Outputs:

Void

Function:

GetTaskId(TaskId)

EnterExclusiveArea()

For each Active CRC Hardware Unit:

If CrcHwIdxSts == Available:

If ResvCrcCall == False:

CrcHwSts[CrcHwIdx].CrcHwSts = Busy

CrcHwSts[CrcHwIdx].TaskId = TaskId

Else:

CrcHwSts[CrcHwIdx].CrcHwSts = Reserve

CrcHwSts[CrcHwIdx].TaskId = ReserveTaskId[CrcHwIdx]

End If

Break For Loop

End If

End For Loop

ExitExclusiveArea()

### Sub-Function: ResvCrcHwUnit

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_48, CM800A\_67, CM800A\_68, CM800A\_70***

This function shall allow components to reserve a single hardware unit to compute a larger CRC utilizing hardware features, such as DMA. The function shall temporarily reserve the hardware unit until released by the component. A reservation key will be provided when an index is reserved. This key will need to be stored in the caller’s RAM as it is used to determine which hardware index to release when called.

Pseudo code notes:

* The function CrcRegConfig() is not a function
  + It represents setting the DCRA register sections ISZ and POL, and setting COUT to the StartValue to meet the desired CRC calculation as desired by CrcConfg.
* LoopCrcHwIdxInReg and LoopCrcHwIdxOutReg are the CIN and COUT for each of the CRC hardware units.

Function Inputs:

In: Mode := 1= CRCHWRESVMOD\_RESV, 0= CRCHWRESVMOD\_RELS

In: CrcConfig := See table below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enumeration | CrcConfig (Enum Value) | DCRAnISZ | DCRAnPOL | Meaning |
| CRCHWRESVCFG\_32BITCRC32BITWIDTH | 0 | 0 | 0 | 32-Bit CRC / 32-Bit Access Width |
| CRCHWRESVCFG\_32BITCRC16BITWIDTH | 1 | 0 | 1 | 32-Bit CRC / 16-Bit Access Width |
| CRCHWRESVCFG\_32BITCRC8BITWIDTH | 2 | 0 | 2 | 32-Bit CRC / 8-Bit Access Width |
| CRCHWRESVCFG\_16BITCRC16BITWIDTH | 3 | 1 | 1 | 16-Bit CRC / 16-Bit Access Width |
| CRCHWRESVCFG\_16BITCRC8BITWIDTH | 4 | 1 | 2 | 16-Bit CRC / 8-Bit Access Width |
| CRCHWRESVCFG\_8BITCRC | 5 | 2 | 2 | 8-Bit CRC / 8-Bit Access Width (SAE-J 1850) |
| CRCHWRESVCFG\_8BITCRCH2F | 6 | 3 | 2 | 8-Bit CRC / 8-Bit Access Width (Polynomial 0x2F) |

Out: CrcHwIdxInReg

Out: CrcHwIdxOutReg

In: StartValue

In/Out: ResvKey

Function Outputs:

Standard Return

* OK = Successful reservation.
* NOT\_OK = Reservation failed, CRC not configured.

Function:

FuncReturn = NOT\_OK

If Mode == Reserve and CrcConfig < 7:

For each Active CRC Hardware Unit:

If CrcHwIdxSts == Available:

GetAvlCrcHwUnit(TRUE)

ResvKey = ReserveTaskId[CrcHwLoopIdx]

CrcResvd = TRUE

CrcHwIdx = CrcHwLoopIdx

Break For Loop

End If

End For Loop

If (CrcResvd == TRUE):

CrcRegConfig(CrcConfig, StartValue)

CrcHwIdxInReg = Address of Selected Index Input Register

CrcHwIdxOutReg = Address of Selected Output Register

FuncReturn = OK

End If

If(FuncReturn == NOT\_OK):

CrcHwIdxInReg = 0

CrcHwIdxOutReg = 0

ResvKey = 0

End If

Else If Mode == Release:

/\* Release path \*/

For each Active CRC Hardware Unit:

If ((CrcHwIdxSts == Reserved) and

(ResvKey = CrcHwSts[CrcHwIdx].Taskid)):

RelsCrcHwUnit(CrcHwIdx)

FuncReturn = OK

Break For Loop

End If

End For Loop

If(FuncReturn == NOT\_OK):

CrcHwIdxInReg = 0

CrcHwIdxOutReg = 0

ResvKey = 0

End If

Else:

FuncReturn = NOT\_OK

CrcHwIdxInReg = 0

CrcHwIdxOutReg = 0

ResvKey = 0

End If

## SyncCrc API Functions

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_64, CM800A\_66***

The functions defined in the following sub sections shall follow the following format and functionally meet the pseudo code.

Function Inputs:

In/Out: DataPtr\_Arg†

In: Len\_Arg

In: StrtVal\_Arg†

In: FirstCall\_Arg

In/Out: CalcCrcRes\_Arg†

† - Argument data type depends on the API function (choices are 8, 16, or 32-bit).

Function Outputs:

Standard Return

* OK = Successful CRC calculation.
* NOT\_OK = CRC Hardware unit could not be reserved, no calculation performed. CalcCrcRes\_Arg shall be set to 0.

Function:

ErrRtn = OK

GetTaskId(TaskId)

GetAvlCrcHwUnit()

For each Active CRC Hardware Unit:

If TaskId == RAMTable.TaskId:

CrcHwIdx = LoopIdx

Break For Loop

End If

End For Loop

If CrcHwIdx != Invalid CRC Index:

CrcRegConfig() /\* Configure Registers for the function type \*/

If FirstCall\_Arg == True:

COUT = Default CRC Init Value

Else:

COUT = StrtVal\_Arg

End If

For 0 to Len\_Arg:

CIN = DataPtr[LenIdx]

End For

CalcCrcRes\_Arg = COUT

RelsCrcHwUnit(CrcHwIdx)

Else:

CalcCrcRes\_Arg = 0

ErrRtn = NOT\_OK

End If

### Sub-Function: 32-Bit Ethernet CRC

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_52, CM800A\_64***

The CRC module shall implement the CRC32 routine based on the IEEE-802.3 CRC32 Ethernet Standard. In the event a CRC cannot be calculated the function shall return a CRC result of 0 and notify the calling software component that the CRC result was not calculated.

#### Hardware Related Design

N/A

#### Software Related Design

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_53, CM800A\_54***

The following table defines the parameters used to define the 32-Bit Ethernet functions.

|  |  |
| --- | --- |
| CRC Result Width | 32 Bits |
| Polynomial | 0x04C11DB7  or  X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 + 1  or |
| Initial Value | 0xFFFFFFFF |
| Input Data Width | 8-Bit / 16-Bit / 32-Bit |
| Input Data Reflected | Yes |
| Result Data Reflected | Yes |
| XOR Value | 0xFFFFFFFF |
| Check | 0xCBF43926 |
| Magic Check | 0xDEBB20E3 |

##### Calc32BitCrc\_u08

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_55***

This function shall implement the algorithm defined in Section 5.2.1.2. The input data shall be accessed in 8-Bit segments. This function shall be used in place of the AUTOSAR Crc\_CalculateCRC32 function and described in section 5.3.

##### Calc32BitCrc\_u16

This function shall implement the algorithm defined in Section 5.2.1.2. The input data shall be accessed in 16-Bit segments. This requires that the input data be aligned for 16-Bit access.

##### Calc32BitCrc\_u32

This function shall implement the algorithm defined in Section 5.2.1.2. The input data shall be accessed in 32-Bit segments. This requires that the input data be aligned for 32-Bit access.

### Sub-Function: 16-Bit CRC

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_56, CM800A\_64***

The CRC module shall implement the CRC16 routine based on the CCITT CRC16 Standard. In the event a CRC cannot be calculated the function shall return a CRC result of 0 and notify the calling software component that the CRC result was not calculated.

#### Hardware Related Design

N/A

#### Software Related Design

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_57, CM800A\_58***

The following table defines the parameters used to define the 16-Bit functions.

|  |  |
| --- | --- |
| CRC Result Width | 16 Bits |
| Polynomial | 0x1021  or  X16 + X12 + X5 + 1  or |
| Initial Value | 0xFFFF |
| Input Data Width | 8-Bit / 16-Bit |
| Input Data Reflected | No |
| Result Data Reflected | No |
| XOR Value | 0x0000 |
| Check | 0x29B1 |
| Magic Check | 0x0000 |

##### Calc16BitCrc\_u08

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_59***

This function shall implement the algorithm defined in Section 5.2.2.2. The input data shall be accessed in 8-Bit segments. This function shall be used in place of the AUTOSAR Crc\_CalculateCRC16() function and described in section 5.3.

##### Calc16BitCrc\_u16

This function shall implement the algorithm defined in Section 5.2.2.2. The input data shall be accessed in 16-Bit segments. This requires that the input data be aligned for 16-Bit access.

### Sub-Function: 8-Bit SAE-J1850 CRC

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_60, CM800A\_64***

The CRC module shall implement the CRC8 routine based on the SAE-J1850 CRC8 Standard. In the event a CRC cannot be calculated the function shall return a CRC result of 0 and notify the calling software component that the CRC result was not calculated.

#### Hardware Related Design

N/A

#### Software Related Design

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_61, CM800A\_62***

The following table defines the parameters used to define the 8-Bit functions.

|  |  |
| --- | --- |
| CRC Result Width | 8 Bits |
| Polynomial | 0x1D  or  X8 + X4 + X3 + X2 + 1  or |
| Initial Value | 0xFF |
| Input Data Width | 8-Bits |
| Input Data Reflected | No |
| Result Data Reflected | No |
| XOR Value | 0xFF |
| Check | 0x4B |
| Magic Check | 0xC4 |

##### Calc8BitCrc

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_63***

This function shall implement the algorithm defined in Section 5.2.3.2. The input data shall be accessed in 8-Bit segments. This function shall be used in place of the AUTOSAR Crc\_CalculateCRC8() function and described in section 5.3.

### Sub-Function: 8-Bit 0x2F CRC

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_60, CM800A\_64***

The CRC module shall implement the CRC8 routine based with a 0x2F polynomial. In the event a CRC cannot be calculated the function shall return a CRC result of 0 and notify the calling software component that the CRC result was not calculated.

#### Hardware Related Design

N/A

#### Software Related Design

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_61, CM800A\_62***

The following table defines the parameters used to define the 8-Bit functions.

|  |  |
| --- | --- |
| CRC Result Width | 8 Bits |
| Polynomial | 0x2F  or  X8 + X5 + X3 + X2 + X1 + 1  or |
| Initial Value | 0xFF |
| Input Data Width | 8-Bit |
| Input Data Reflected | No |
| Result Data Reflected | No |
| XOR Value | 0xFF |
| Check | 0xDF |
| Magic Check | 0x42 |

##### Calc8BitCrc0X2F

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_63***

This function shall implement the algorithm defined in Section 5.2.4.2. The input data shall be accessed in 8-Bit segments. This function shall be used in place of the AUTOSAR Crc\_CalculateCRC8H2F() function and described in section 5.3.

## Sub-Function: AUTOSAR API Wrapper

***#REQ: The following requirement(s) are met by the design feature below: Requirement ID: CM800A\_51***

The following API wrapper functions shall be to replace the AUTOSAR CRC API libraries for software CRC calculations. The return value of OK/NOT\_OK from the SyncCrc functions are ignored as AUTOSAR does not provide this error handling in their requirements.

Function Inputs:

In/Out: DataPtr\_Arg

In: Len\_Arg

In: StrtVal\_Arg†

In: FirstCall\_Arg

† - Argument data type depends on the API function (8, 16, or 32-bit).

Function Outputs:

CrcRes†: Result of the CRC calculation.

† - Argument data type depends on the API function.

### Sub-Function: Crc\_CalculateCRC32

Function:

Calc32BitCrc\_u08\_Oper( DataPtr\_Arg, Len\_Arg, StrtVal\_Arg,

FirstCall\_Arg, &CrcRes)

Return CrcRes

### Sub-Function: Crc\_CalculateCRC16

Function:

Calc16BitCrc\_u08\_Oper( DataPtr\_Arg, Len\_Arg, StrtVal\_Arg,

FirstCall\_Arg, &CrcRes)

Return CrcRes

### Sub-Function: Crc\_CalculateCRC8

Function:

Calc8BitCrc\_Oper ( DataPtr\_Arg, Len\_Arg, StrtVal\_Arg,

FirstCall\_Arg, &CrcRes)

Return CrcRes

### Sub-Function: Crc\_CalculateCRC8H2F

Function:

Calc8BitCrc0X2F\_Oper ( DataPtr\_Arg, Len\_Arg, StrtVal\_Arg,

FirstCall\_Arg, &CrcRes)

Return CrcRes

# Timing / Execution Constraints

## Rationale / Comments

The functions defined in this document are synchronous functions and are not required to be scheduled to run at a periodic rate.

## Rates and State Execution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sub-Function Name** | **Rate (ms)** | **Cold Init** | **Warm Init** | **Operate** | **Disable** |
| SyncCrcInit0 | N/A | Y\* | Do Not Execute | Do Not Execute | Do Not Execute |
| SyncCrcInit1 | N/A | Do Not Execute | Y | Do Not Execute | Do Not Execute |
| Calc32BitCrc\_u08 | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc32BitCrc\_u16 | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc32BitCrc\_u32 | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc16BitCrc\_u08 | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc16BitCrc\_u16 | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc8BitCrc0X2F | N/A | Do Not Execute | N/A | N/A | N/A |
| Calc8BitCrc | N/A | Do Not Execute | N/A | N/A | N/A |
| Crc\_CalculateCRC32 | N/A | Do Not Execute | N/A | N/A | N/A |
| Crc\_CalculateCRC16 | N/A | Do Not Execute | N/A | N/A | N/A |
| Crc\_CalculateCRC8 | N/A | Do Not Execute | N/A | N/A | N/A |
| Crc\_CalculateCRC8H2F | N/A | Do Not Execute | N/A | N/A | N/A |

Y\* -- No CRC calculations can be performed with this component before this function is executed.

# Serial Communications Interfaces

N/A

# Additional Information

N/A

# Revision Record & Change Approval

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rev** | **Date** | **Change Control #** | **Drw** | **Change Description** |
| 01.00.00 | 12-Jan-16 | EA4#1843 | KJS | Initial Release of this document |