*Technion*

*Electrical Engineering Department*

High Speed Digital System Lab

Message Pack

And Checksum

Documentation

**Version**: 1.0

**Last update**: 26.02.2011

**Written By**: Beeri Schreiber and Alon Yavich

Table of content

[Table of Changes 3](#_Toc293164407)

[1 Scope 3](#_Toc293164408)

[2 Abbreviations 3](#_Toc293164409)

[3 General Description 4](#_Toc293164410)

[3.1 Message Pack Decoder 4](#_Toc293164411)

[3.2 Message Pack Encoder 4](#_Toc293164412)

[3.3 Checksum 5](#_Toc293164413)

[4 Message Pack Decoder 6](#_Toc293164414)

[4.1 Resources 6](#_Toc293164415)

[4.2 Message Pack Decoder Pinout 6](#_Toc293164416)

[4.3 Message Pack Decoder Generic Parameters 7](#_Toc293164417)

[4.4 Message Pack Decoder Wave 8](#_Toc293164418)

[4.4.1 Data from UART Generator 8](#_Toc293164419)

[4.4.2 RAM Handshake 9](#_Toc293164420)

[4.4.3 CRC Handshake 9](#_Toc293164421)

[4.4.4 Output Registers 9](#_Toc293164422)

[4.4.5 Error Flags 9](#_Toc293164423)

[4.4.6 Message Pack Decoder Done 9](#_Toc293164424)

[4.5 Message Pack Decoder Wave – CRC and EOF Error 9](#_Toc293164425)

[4.6 Message Pack Decoder State Machine 10](#_Toc293164426)

[5 Message Pack Encoder 11](#_Toc293164427)

[5.1 Resources 11](#_Toc293164428)

[5.2 Message Pack Encoder Pinout 11](#_Toc293164429)

[5.3 Message Pack Encoder Generic Parameters 12](#_Toc293164430)

[5.4 Message Pack Encoder Wave 13](#_Toc293164431)

[5.4.1 Input Registers 13](#_Toc293164432)

[5.4.2 FIFO Handshake 13](#_Toc293164433)

[5.4.3 CRC Handshake 15](#_Toc293164434)

[5.4.4 RAM Handshake 15](#_Toc293164435)

[5.4.5 Message Pack Encoder Done 15](#_Toc293164436)

[5.5 Message Pack Encoder State Machine 15](#_Toc293164437)

[6 Checksum Block 16](#_Toc293164438)

[6.1 Resources 16](#_Toc293164439)

[6.2 Checksum Pinout 16](#_Toc293164440)

[6.3 Checksum Generic Parameters 16](#_Toc293164441)

[6.4 Checksum Handshake 17](#_Toc293164442)

[6.4.1 Signed / Unsigned Checksum 17](#_Toc293164443)

[6.4.2 Output Checksum Width 18](#_Toc293164444)

[6.5 Checksum Wave 18](#_Toc293164445)

[6.6 Checksum RTL 18](#_Toc293164446)

[7 Performed Tests for Message Pack System 19](#_Toc293164447)

Table of Tables

[Table 1 – Table of Changes 3](#_Toc286564228)

[Table 2 – Message Pack Decoder Pinout 7](#_Toc286564229)

[Table 3 – Message Pack Decoder Generic Parameters 8](#_Toc286564230)

[Table 4 – Message Pack Encoder Pinout 12](#_Toc286564231)

[Table 5 – Message Pack Encoder Generic Parameters 13](#_Toc286564232)

[Table 6 – Checksum Pinout 16](#_Toc286564233)

[Table 7 – Checksum Generic Parameters 17](#_Toc286564234)

Tables of Figures

[Figure 1 – Message Pack Structure 4](#_Toc286564235)

[Figure 2 – Message Pack Decoder Pinout 6](#_Toc286564236)

[Figure 3 – Message Pack Decoder and Encoder Wave 8](#_Toc286564237)

[Figure 4 – Message Pack Decoder - CRC and EOF Error 9](#_Toc286564238)

[Figure 5 – Message Pack Decoder State Machine 10](#_Toc286564239)

[Figure 6 – Message Pack Enccoder Pinout 11](#_Toc286564240)

[Figure 7 – Message Pack Encoder Wave 13](#_Toc286564241)

[Figure 8 – Message Pack Encoder <--> FIFO Handshake 14](#_Toc286564242)

[Figure 9 – Message Pack Encoder State Machine 15](#_Toc286564243)

[Figure 10 – Checksum Pinout 16](#_Toc286564244)

[Figure 11 – Checksum Handshake (1) 17](#_Toc286564245)

[Figure 12 – Checksum Handshake (2) 17](#_Toc286564246)

[Figure 13 – Checksum Wave 18](#_Toc286564247)

[Figure 14 – Checksum RTL 18](#_Toc286564248)

# Table of Changes

|  |  |  |
| --- | --- | --- |
| Version | Date of Change | Description |
| 1.0 | 26.02.2011 | Creation of documentation |

Table 1 – Table of Changes

# Scope

This document aims to describe the working method of the following IPs:

1. Message Pack Decoder
2. Message Pack Encoder
3. Checksum Block

# Abbreviations

1. MP – Message Pack
2. CRC – Cyclic Redundancy Check
3. SOF – Start of Frame
4. EOF – End of Frame
5. UART – Universal Asynchronous Receiver / Transmitter
6. TB – Test bench

# General Description

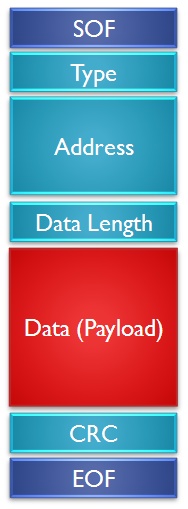


Figure 1 – Message Pack Structure

## Message Pack Decoder

Message Pack Decoder receives a message pack, built from the following blocks:

1. **SOF** – Start of Frame
2. **Type** – Message type
3. **Address** – Address for the data, or for other purposes
4. **Length** – Data length. Data length CANNOT be less than 1.
5. **Data** **(Payload)** – The wrapped data
6. **CRC** – CRC of Type, Address, Length and Data blocks
7. **EOF** – End of Frame

The MP sniffs the data line, until is receives SOF block. Then it decodes the message: Type, Address and Length will be stored into registers, and will be valid when the EOF is received. Data will be stored into RAM.

Special problematic SOF words are being handled:

Suppose SOF = 0xAABBCC.

A message of 0xAABBAABBCC... will be decode correctly by the MP decoder.

## Message Pack Encoder

Message Pack Decoder Encoder transmits data from the Type and Address registers, and from the RAM, in a Message Pack format:

1. **SOF** – Start of Frame
2. **Type** – Message type
3. **Address** – Address for the data, or for other purposes
4. **Length** – Data length. Data length CANNOT be less than 1.
5. **Data (Payload)** – The wrapped data
6. **CRC** – CRC of Type, Address, Length and Data blocks
7. **EOF** – End of Frame

## Checksum

The Checksum receives data from the Message Pack Decoder / Encoder, and calculates the Checksum for the received data. It is possible to define greater output length than the input length.

**IMPORTAT**: Message Pack uses CRC block. In this project Checksum replaces the CRC block.

# Message Pack Decoder

## Resources

Required resources, when synthesizing, using Quartus, for Altera's Cyclon II FPGA:

* 4 AND gates
* 2 OR gates
* 179 DFF
* 1 State Machine
* 154 MUX
* 1 Addition Operator
* 1 'Less Than' Operator
* 2 Selector Operator
* 5 Equal Operator

**Maximum Working Frequency**: 180MHz

## Message Pack Decoder Pinout

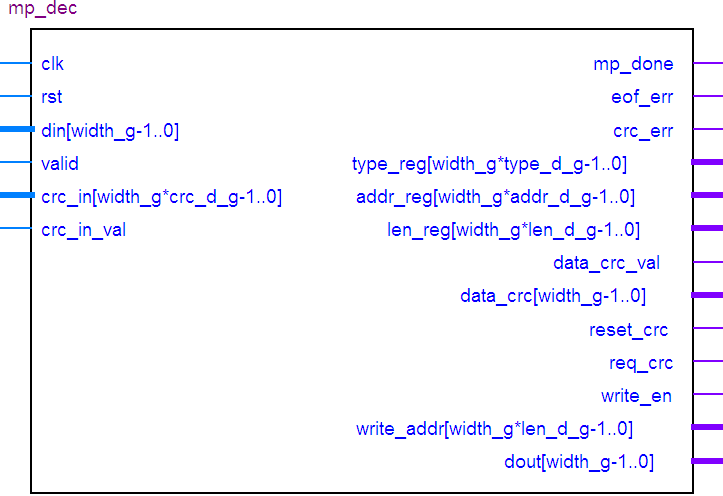


Figure 2 – Message Pack Decoder Pinout

| Pin Name | Direction | Description |
| --- | --- | --- |
| Clk | In | Clock |
| Rst | In | Reset. Reset polarity will be set according to the generic parameter 'reset\_polarity\_g' |
| Din [width\_g-1..0] | In | Input Data from data provider |
| Valid | In | Input data is valid signal |
| Crc\_in [width\_g \* crc\_d\_g-1..0] | In | Input CRC from CRC Block |
| Crc\_in\_val | In | Input CRC is valid signal |
| Mp\_dec\_done | Out | Message Pack Decoder transmission is done |
| Eof\_err | Out | EOF Error – Defined EOF and received EOF are not the same |
| Crc\_err | Out | CRC Error – Calculated and received CRC are not the same |
| Type\_reg [width\_g \* type\_d\_g-1..0] | Out | Output decoded Type. Will be valid together with the *mp\_dec\_done* signal |
| Addr\_reg [width\_g \* addr\_d\_g-1..0] | Out | Output decoded Address. Will be valid together with the *mp\_dec\_done* signal |
| Len\_reg [width\_g \* len\_d\_g-1..0] | Out | Output decoded Data Length. Will be valid together with the *mp\_dec\_done* signal |
| Data\_crc\_val | Out | Data for CRC calculation is valid |
| Data\_crc [width\_g-1..0] | Out | Data to CRC block |
| Reset\_crc | Out | Reset the CRC value |
| Req\_crc | Out | Request for CRC calculated value |
| Write\_en | Out | Write Enable to the RAM |
| Write\_addr [width\_g \* len\_d\_g-1..0] | Out | Write Address to the RAM |
| Dout [width\_g-1..0] | Out | Output Data |

Table 2 – Message Pack Decoder Pinout

## Message Pack Decoder Generic Parameters

| Generic Parameter | Default Value | Description |
| --- | --- | --- |
| Reset\_polartiy\_g | '0' | Reset active in this polarity |
| Len\_dec1\_g | true | TRUE to receive decreased length by 1. For example: in case actual length is 6, 5 will be received. |
| Sof\_d\_g | 1 | SOF block depth |
| Type\_d\_g | 1 | Type block depth |
| Addr\_d\_g | 3 | Address block depth |
| Len\_d\_g | 2 | Length block depth |
| Crc\_d\_g | 1 | CRC block depth |
| Eof\_d\_g | 1 | EOF block depth |
| Sof\_val\_g | 100 | Initial SOF value (decimal = 64hex) |
| Eof\_val\_g | 200 | Initial EOF value (decimal = C8hex) |
| Width\_g | 8 | Data width (number of bits) |

Table 3 – Message Pack Decoder Generic Parameters

## Message Pack Decoder Wave

Figure 3 – Message Pack Decoder and Encoder Wave

The wave is divided into 6 sections:

### Data from UART Generator

The UART generator generates UART transmission, which is NOT relevant for the MP blocks. MP Decoder receives the data from *din*, together with the *valid* signal.

When SOF (64hex in this example) is being received – the message decoding process is initialized.

### RAM Handshake

MP Decoder transmits the received payload into the RAM. Data is being transmitted, together with the RAM address and valid signal

### CRC Handshake

MP Decoder transmits the received Type, Address, Length and Payload data to the CRC block. See Checksum description for handshake explanation.

### Output Registers

When correct EOF is received, *mp\_dec\_done* flag will be raised. Type, Address and Length will be available form that point.

### Error Flags

There are two error output flags:

1. **CRC Error** – will be raised in case received CRC and calculated CRC are not equal.
2. **EOF Error** – will be raised in case received EOF and defined EOF (by generic parameter) are not equal. In case such error has occurred – *mp\_dec\_done* flag will not be raised.

### Message Pack Decoder Done

When correct EOF is received, *mp\_dec\_done* flag will be raised. In case of EOF error – this flag will not be received.

## Message Pack Decoder Wave – CRC and EOF Error

Figure 4 – Message Pack Decoder - CRC and EOF Error

In the above figure, transmitted CRC is 4Dhex, while the calculated CRC is 4Chex. Note that the CRC error and General error flags are raised.

Transmitted EOF is C9hex, while the defined EOF, by generic parameter, is C8hex. Note that the EOF Error flag is raised, and *mp\_dec\_done* is not being raised.

## Message Pack Decoder State Machine

Figure 5 – Message Pack Decoder State Machine

# Message Pack Encoder

## Resources

Required resources, when synthesizing, using Quartus, for Altera's Cyclon II FPGA:

* 3 AND gates
* 8 OR gates
* 129 DFF
* 1 State Machine
* 228 MUX
* 2 Addition Operator
* 1 'Less Than' Operator
* 61 Selector Operator
* 3 Equal Operator

**Maximum Working Frequency**: 180MHz

## Message Pack Encoder Pinout

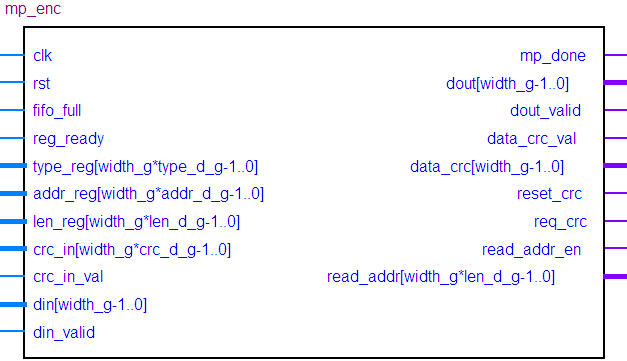


Figure 6 – Message Pack Enccoder Pinout

| Pin Name | Direction | Description |
| --- | --- | --- |
| Clk | In | Clock |
| Rst | In | Reset. Reset polarity will be set according to the generic parameter 'reset\_polarity\_g' |
| Fifo\_full | In | FIFO is full, an cannot receive more data from MP Encoder |
| Reg\_ready | In | Input Type, Address and Data Length registers values are ready |
| Type\_reg [width\_g \* type\_d\_g-1..0] | In | Input Type value. Will be valid together with the *reg\_ready* signal |
| Addr\_reg [width\_g \* addr\_d\_g-1..0] | In | Input Address value. Will be valid together with the *reg\_ready* signal |
| Len\_reg [width\_g \* len\_d\_g-1..0] | In | Input Data Length value. Will be valid together with the *reg\_ready* signal |
| Crc\_in [width\_g \* crc\_d\_g-1..0] | In | Calculated CRC value from Checksum block |
| Crc\_in\_val | In | Calculated CRC value from Checksum block is valid |
| Din [width\_g-1..0] | In | Input data (payload), from RAM |
| Din\_valid | In | Input data (payload), from RAM is valid |
| Mp\_done | Out | Message Pack has been successfully transmitted. This flag will be raised together with the EOF output data |
| Dout [width\_g-1..0] | Out | Output data, to the FIFO |
| Dout\_valid | Out | Output data, to the FIFO, is valid |
| Data\_crc\_val | Out | Data to the CRC block is valid |
| Data\_crc [width\_g-1..0] | Out | Data to the CRC, for CRC calculation |
| Reset\_crc | Out | Reset the CRC value to its default value |
| Req\_crc | Out | Request for calculated CRC value |
| Read\_addr\_en | Out | Address to RAM is valid |
| Read\_addr [width\_g \* len\_d\_g-1..0] | Out | Address to RAM |

Table 4 – Message Pack Encoder Pinout

## Message Pack Encoder Generic Parameters

| Generic Parameter | Default Value | Description |
| --- | --- | --- |
| Reset\_polartiy\_g | '0' | Reset active in this polarity |
| Len\_dec1\_g | true | TRUE to receive decreased length by 1. For example: in case actual length is 6, 5 will be received. |
| Sof\_d\_g | 1 | SOF block depth |
| Type\_d\_g | 1 | Type block depth |
| Addr\_d\_g | 3 | Address block depth |
| Len\_d\_g | 2 | Length block depth |
| Crc\_d\_g | 1 | CRC block depth |
| Eof\_d\_g | 1 | EOF block depth |
| Sof\_val\_g | 100 | Initial SOF value (decimal = 64hex) |
| Eof\_val\_g | 200 | Initial EOF value (decimal = C8hex) |
| Width\_g | 8 | Data width (number of bits) |

Table 5 – Message Pack Encoder Generic Parameters

## Message Pack Encoder Wave

Figure 7 – Message Pack Encoder Wave

The wave is divided into 5 sections:

### Input Registers

Input registers of Type, Address and Length.

### FIFO Handshake

MP Encoder sends all the blocks data to the FIFO. From the FIFO, message will be transmitted by some data transmitter.

In case FIFO is full – MP Encoder will pause the transmission, and will resume the transmission as soon as the FIFO will not be full.

When the FIFO is not FULL, MP Encoder can transmit data into the FIFO, together with the *enc\_dout\_val* flag.

Figure 8 – Message Pack Encoder <--> FIFO Handshake

### CRC Handshake

MP Encoder transmits Type, Address, Length and Payload data to the CRC block. See Checksum description for handshake explanation.

### RAM Handshake

MP Encoder reads the payload data, to be transmitted, from the RAM. Data is being read, together with the RAM address and read enable signal. Output data from RAM is companioned with valid signal.

### Message Pack Encoder Done

*mp\_enc\_done* flag will be raised when all data has been transmitted to the FIFO.

## Message Pack Encoder State Machine

Figure 9 – Message Pack Encoder State Machine

# Checksum Block

## Resources

Required resources, when synthesizing, using Quartus, for Altera's Cyclon II FPGA:

* 9 DFF
* 24 MUX
* 1 Addition Operator

**Maximum Working Frequency**: 290MHz

## Checksum Pinout

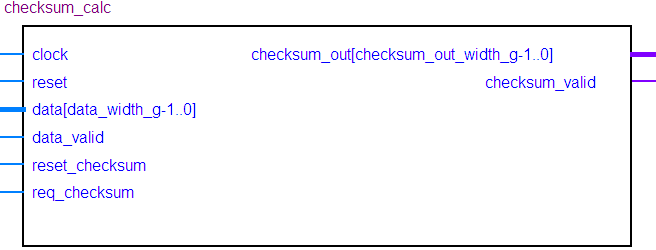


Figure 10 – Checksum Pinout

| Pin Name | Direction | Description |
| --- | --- | --- |
| Clock | In | Clock |
| Reset | In | Reset. Reset polarity will be set according to the generic parameter 'reset\_polarity\_g' |
| Data [data\_width\_g-1..0] | In | Input data, to be calculated by Checksum |
| Data\_valid | In | Input data for calculation is valid |
| Reset\_Checksum | In | Reset Checksum to its default value |
| Req\_Checksum | In | Request for Checksum value |
| Checksum\_out [Checksum\_out\_width\_g-1..0] | out | Calculated Checksum value |
| Checksum\_valid | out | Calculated Checksum value is valid |

Table 6 – Checksum Pinout

## Checksum Generic Parameters

| Generic Parameter | Default Value | Description |
| --- | --- | --- |
| Reset\_polartiy\_g | '0' | Reset active in this polarity |
| Signed\_Checksum\_g | False | TRUE for signed Checksum, FALSE for unsigned Checksum |
| Checksum\_init\_val\_g | 0 | Default natural value, when resetting Checksum |
| Checksum\_out\_width\_g | 8 | Output data width (number of bits). This parameter's value may be equal or greater than *data\_width\_g* |
| Data\_width\_g | 8 | Input data width (number of bits), for Checksum calculation |

Table 7 – Checksum Generic Parameters

## Checksum Handshake

**Option 1** – *req\_checksum* rise at the same clock with the *data\_valid* assertion:

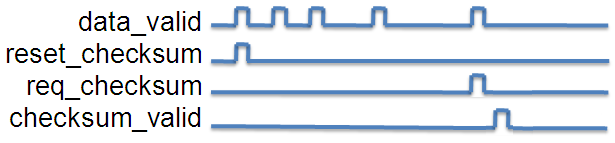


Figure 11 – Checksum Handshake (1)

**Option 2** – *req\_checksum* rise one clock (or more) after the *data\_valid* assertion:

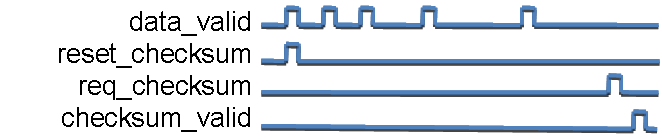


Figure 12 – Checksum Handshake (2)

**Reset Checksum –** Reset the Checksum to the default value, defined by the generic parameter *Checksum\_init\_val\_g*. First *data\_valid* signal can be activated together with the *reset\_Checksum* signal or one (or more) clock after the *reset\_Checksum* signal activation,

**Request for Checksum** – Data will be valid (*Checksum\_valid* activation) one clock after activation of *req\_Checksum*.

### Signed / Unsigned Checksum

Checksum can be calculated both in signed / unsigned modes. Note that in case of signed Checksum, the MSB bit is reserved as the sign bit, which means that the input and output data width will be sized *(data\_width\_g - 1*).

### Output Checksum Width

Output Checksum width, set by the generic parameter *Checksum\_out\_width\_g*, may be greater or equal to input Checksum width, set by the generic parameter *data\_width\_g*.

## Checksum Wave

The following wave demonstrates Unsigned Checksum:

Figure 13 – Checksum Wave

## Checksum RTL

Figure 14 – Checksum RTL

# Performed Tests for Message Pack System

The following tests were performed on the **Message Pack** system, using UART TX and RX:

1. Transmission of problematic SOF words. For example: Suppose SOF = 0xAABBCC. A message of 0xAABBAABBCC... will be decode correctly by the MP decoder.
2. Different sizes of SOF (8 bits, 16 bits, 24 bits)
3. Different sizes of EOF (8 bits, 16 bits)
4. Different sizes of CRC block (8 bits, 16 bits)
5. Different data bytes transmission (5 bytes, 6 bytes, 1024 bytes where tested) 🡪 Different length block.
6. Length is decreased by 1, and Length, with actual length (*len\_dec1\_g* = true / false)
7. Transmission of incorrect CRC
8. Transmission of incorrect EOF
9. Transmission of special words, such as 1024 times 0x00 and 1024 times 0xFF.
10. Transmission of exactly 256 words (length = 255, when *len\_dec1\_g* = true)

The following tests were performed on the **Checksum block**, using individual Test Bench:

1. Signed and Unsigned Checksum
2. Checksum output equal and greater than input data
3. Request for checksum value (*req\_checksum* signal) at the same clock that the last data checksum output (*data\_valid* signal) is valid
4. Request for checksum value (*req\_checksum* signal) one clock after the last data checksum output (*data\_valid* signal) is valid
5. Different initial values (0x00, 0xFF)