Mailbox Manager Datasheet MBM V 1.1

Resources	PSoC® Blocks				API Memory (Bytes)		Pins
	Digital	Analog CT	Analog SC	Timer	Flash	RAM	
All PSoC Devices							
	0	0	0	0	56	6	0
Every next Mailbox member variable	-	-	-	-	0	+variable size	-

Features and Overview

- Manage an area of RAM (the mailbox) to store data that is transferred between one or more client user modules and a communication interface
- Software only
- Define multiple mailboxes
- Presents a register map displayed in tabular form and allows you to select the content and ordering of each mailbox's data

The Mail Box Manager (MBM) User Module provides a transparent protocol layer for custom protocols required by your application. PSoC Designer uses the mailbox protocol for on chip tuning of CapSense® applications. The MBM User Module supplies an interface between communication user modules and client user modules. For example:

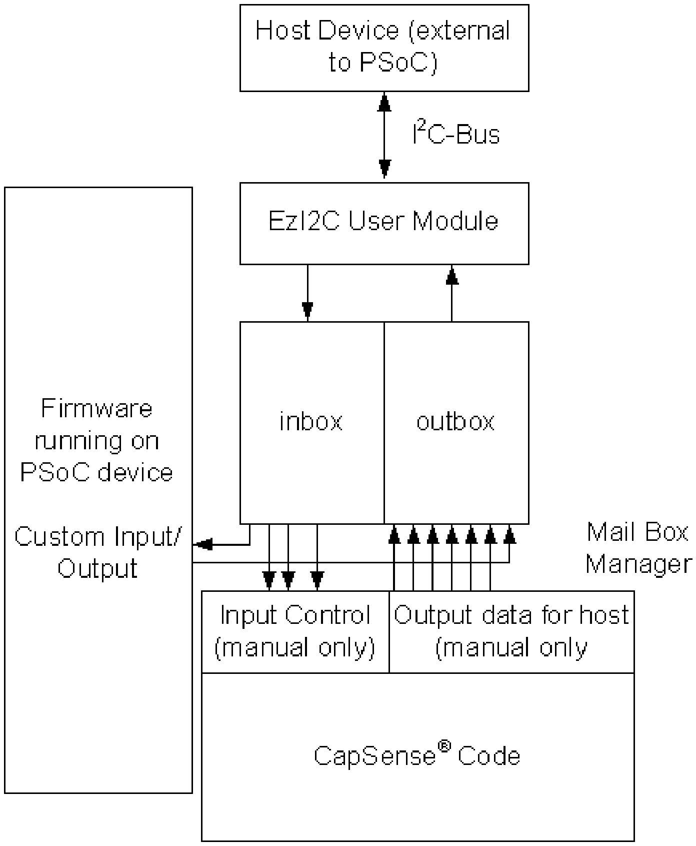
- A sensor interface user module measures a sensors and outputs data, such as CSD; this type of user module is referred to throughout this document as the "client".
- A communication interface controls a HW or SW communication block, such as EzI2C.

The Mailbox Manager (MBM) serves as an interface between these two blocks, providing a way to organize and synchronize the data exchanged between them. Your firmware can access the mailbox for any use. The MBM User Module only supports custom use. In other words, all client user module data must be manually copied to the mailbox structure by your code.

The following block diagram shows the logical interconnection of the three blocks in a sample system.

The MBM User Module supports multiple client user modules but only a single communication interface. In addition, the association between the user module and the communication interface is established manually.

Figure 1. MBM Logical Interconnection Block Diagram



You may need different modes to operate your project. The most common situation is when you need one mode of operation for production and another for tuning your system. The former register map is smaller and only contains the data required by the end system, while the latter is larger and contains extra data useful in analyzing and tuning the system performance during the design phase. For this reason, the MBM User Module supports multiple register map views.

This allows you to easily select between different register maps by changing a GUI selection and rebuilding your project. Each of your mailboxes may have different register map views, giving complete flexibility as the parts of your design you would like to use.

Quick Start

The following steps represent a typical use model for the MBM User Module:

1. Place any client user modules (user modules that will publish or consume data using the MBM User Module) such as CapSense or Touchscreen:

   Configure the client user modules and as you configure them, be aware fo the data it requires.

2. Place the MBM User Module.

3. Configure the MBM User Module:
   a) Open the MBM wizard.
   b) Define the number of mailboxes.
   c) Define name for each mailbox. Also configure the main and tuning register maps.
4. Place the EzI2C User Module.
5. Configure EzI2C to point to the MBM space.
6. Code the rest of project.
7. Tune your project.
   a) Build with tuning enabled on the client user module.
   b) Run the program, invoking client user module tuner.
   c) Adjust settings until acceptable.
   d) Save optimal client user module settings back to the project.
8. Finalize Project
   a) Build with tuning disabled on client user module.
   b) Test the final prototype system.

Functional Description

The MBM User Module is software only. The APIs associated with this user module are entirely contained in the client user modules. The resulting data structure is shared by both the MBM User Module and the client user module. The structure is not defined twice.

The software consists of two main parts: the global mailbox settings and the configuration tools for each individual mailbox.

Figures 2 and 3 show this data flow:

Figure 2. MBM Tuning Data Flow Diagram

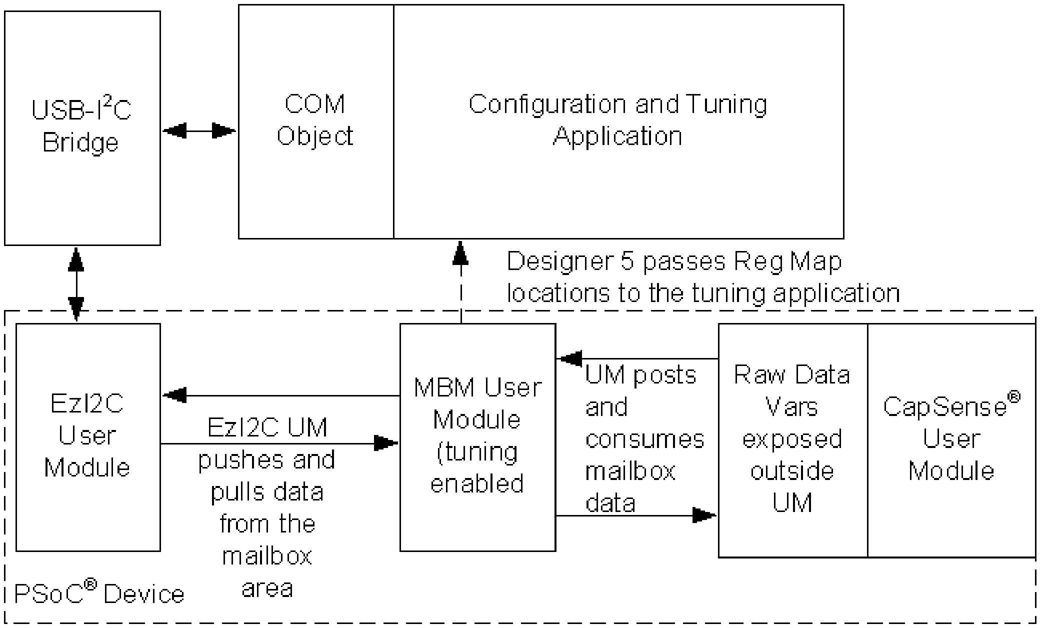
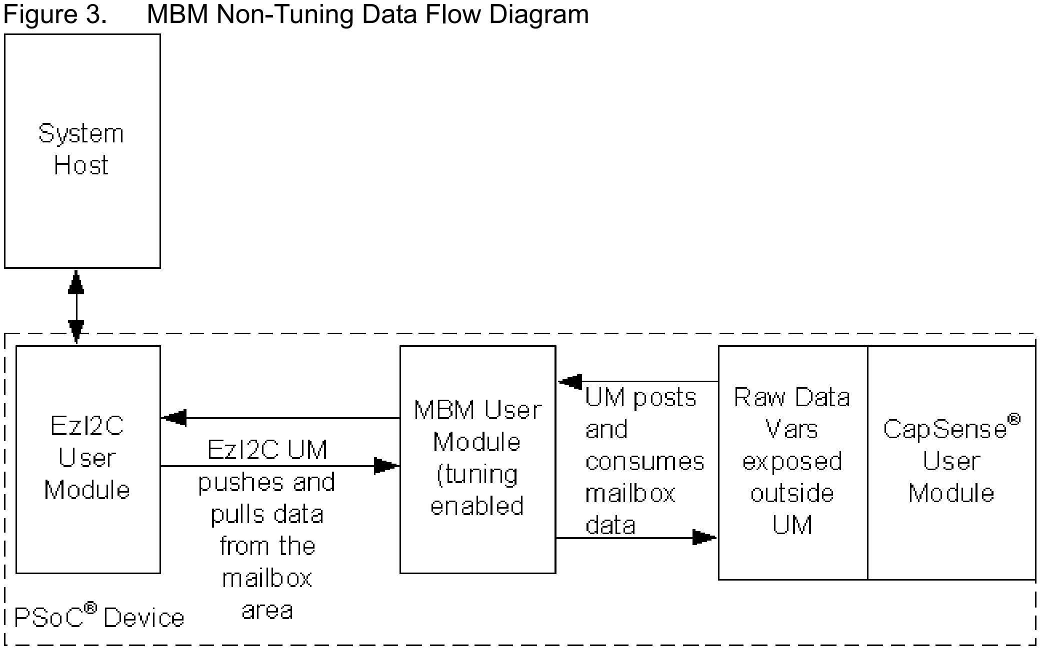


Figure 3. MBM Non-Tuning Data Flow Diagram



Placement

The MBM User Module is software only. It consumes flash and RAM on the target device only. It does not consume any other resources.

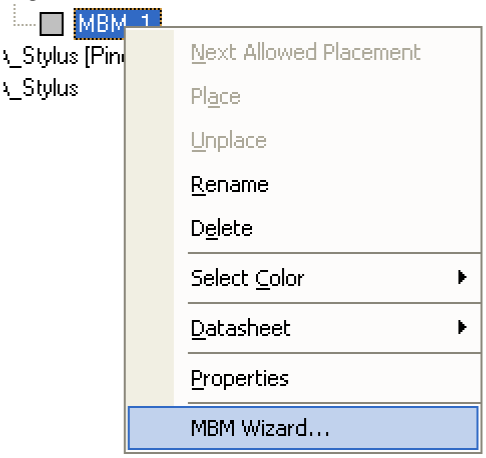
Parameters and Resources

The MBM User Module is completely configured using the Wizard, and does not have any setable Properties in the PSoC Designer Properties window.

Wizard

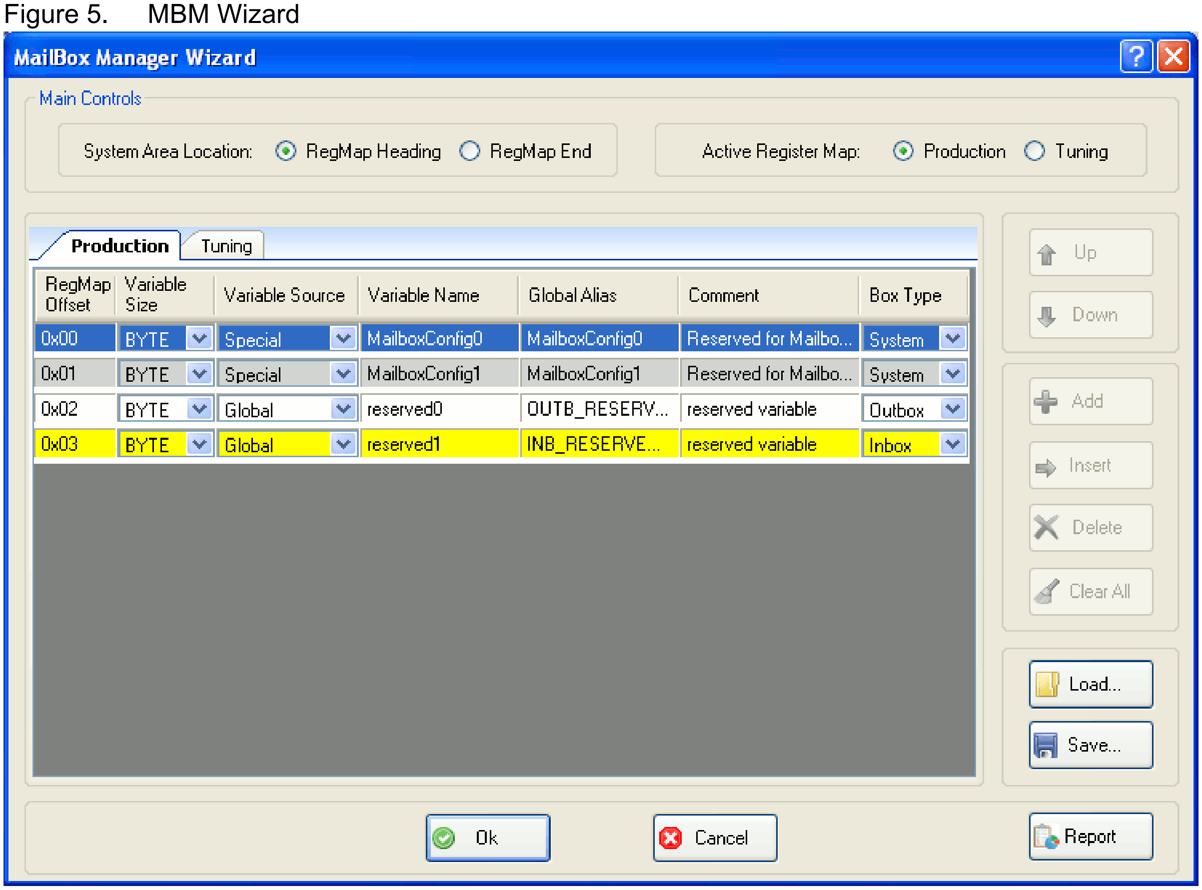
To access the Wizard, right click the MBM User Module, then select the MBM Wizard from the menu.

Figure 4. Wizard Access



The Wizard presents a register map displayed in tabular form and allows you to change the content and ordering of each mailbox's data. The MBM Wizard generates code to map the register map entries to the global alias variables defined in the mailbox configuration tool.

Figure 5. MBM Wizard



An example of MBM wizard is shown in Figure 5. This wizard has the following parameters and controls:

System Area Location

Defines the position of the service variables (MailboxConfig0 and MailboxConfig1) in the mailbox structure.

Active Register Map

You can define two register maps, one for your production device and one with extra values that help you tune and debug your application during development. This selects which of the two is active.

Production

Tuning

You can define two register maps, one for your production device and one with extra values that help you tune and debug your application during development. The two tabs allow you to configure the separate maps.

RegMap Offset

The offset is calculated by the wizard and is not user editable.

Variable Size

Choose an appropriate data size for the variable:

Table 1. Variable Sizes

Variable	Size
Byte	1 byte
Short	2 bytes
Long	4 bytes

Variable Source

Choose Global or RESERVED as the source of the variable.

Variable Name

Enter a valid C variable name for the variable.

Note The MBM Wizard checks to make sure the variable name is valid, but it does not check to see if the variable already exists.

Global Alias

Define a global alias that will be available in your code outside the user module.

Comment

Provide a comment, if desired.

Box Type

Defines the mailbox type which this variable belongs to.

Note Production and Tuning tabs contain two raw variables named MailboxConfig0 and MailboxConfig1. These variables are reserved for the MBM User Module and are not editable. The **Up**, **Down**, **Add**, **Insert**, and **Delete** buttons are not active when these variables are selected.

Up

Down

These buttons allow you to change the order of the variables, moving them up and down within the structure.

Note The RAM offset is dynamically recalculated when rows are moved.

Add

Creates a blank row for a new variable configuration as the last row.

Insert

Inserts a blank row for new variable above the selected row.

Delete

Deletes the selected variable row.

Clear All

Clears all user added rows.

Save

Saves the production and tuning register maps as an *.mbm file.

Load

Loads a previously saved mailbox configuration (.mbm file). Choose **Yes** to load the previously stored mailbox configuration from this PSoC Designer project. Choosing **No** allows you to search for configuration files that are not present in current configuration.

Note While loading previously stored mailbox configuration “Importing mailbox setup” window appears. By choosing “Yes” previously stored mailbox configuration is loaded. Choosing “No” inserts mailboxes which are not present in current configuration. And choosing “Cancel” discards loading configuration

OK

Saves the current mailbox configuration if there are no errors.

Cancel

Exits from the wizard.

Report

Creates a report of the configured mailbox.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the "include" files.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to perserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

MBM_InitializeMailbox

Descriptions:

Initialize the mailbox in the same way as performed by the existing TrueTouch mailbox scheme.

C Prototype:

```
void MBM_InitializeMailbox(void)
```

Assembly:

```
lcall _MBM_InitializeMailbox
```

Parameters

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

MBM_SetBusyFlag

Description:

This function sets the busy flag. It is used in the customer's code that posts messages to the mailbox.

C Prototype:

```
void MBM_SetBusyFlag(void)
```

Assembly:

```
lcall _MBM_SetBusyFlag
```

Parameters

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

MBM_ClearBusyFlag

Description:

This function clears the busy flag. It is used in the customer code that posts messages to the mailbox.

C Prototype:

```
void MBM_ClearBusyFlag(void)
```

Assembly:

```
lcall _MBM_ClearBusyFlag
```

Parameters

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

MBM_CheckDirtyFlag

Description:

This function returns the state of the Dirty Flag. It is used in the customer code that consumes messages from the mailbox.

C Prototype:

```
BYTE MBM_CheckDirtyFlag(void)
```

Assembly:

```
lcall _MBM_CheckDirtyFlag
```

Parameters

None

Return Value:

BYTE value - the state of the Dirty Flag.

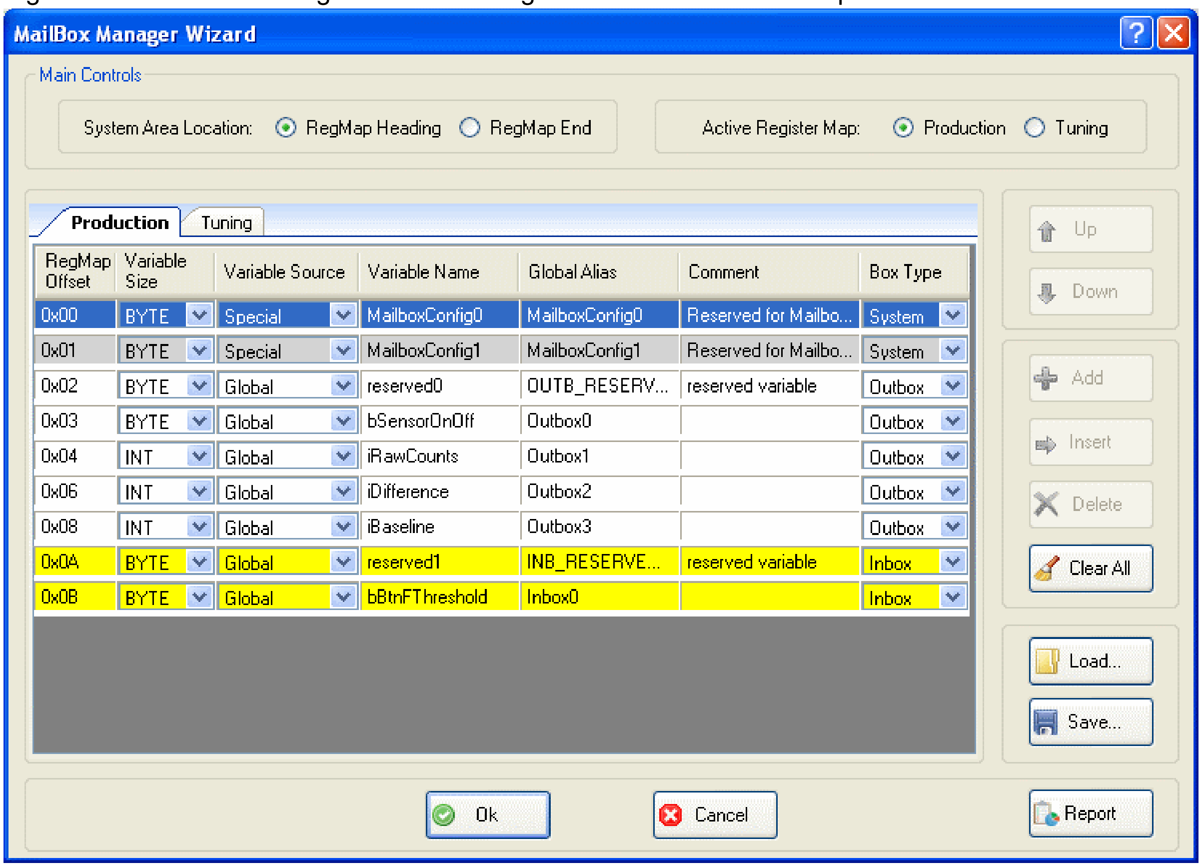
Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

The C example illustrated here shows you how to use the MBM User Module for parts that have CSD User Module. In this example, CSD is used as the client user module. The communication with the CSD module is performed with the System Host through the EZI2C User Module. The MailBox Manager Wizard settings used for this code example are shown in Figure 6:

Figure 6. MailBox Manager Wizard Settings for the First Code Example

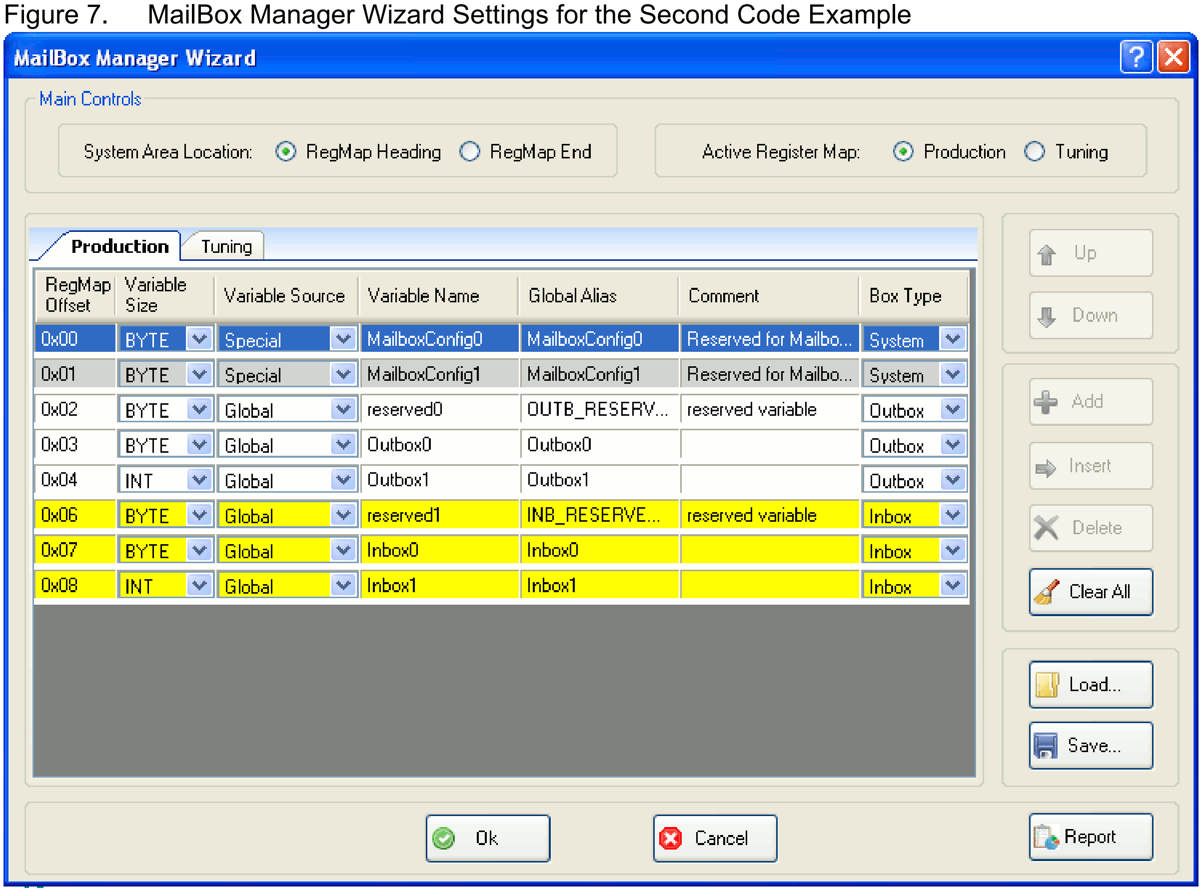



```
//------------------------------------------------------------------------
// Sample C code for the MBM User Module
//------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

void main(void)
{
    M8C_EnableGInt;
MBM_InitializeMailbox();

CSD_Start();
CSD_InitializeBaselines() ; //scan all sensors first time, init baseline
CSD_SetDefaultFingerThresholds() ;

EzI2Cs_SetRamBuffer(sizeof(MBM_MyMailbox), 4, (BYTE *) &MBM_MyMailbox);
    EzI2Cs_Start();                         // Turn on I2C
    EzI2Cs_EnableInt();
```

```
while (1) {
   CSD_ScanAllSensors(); //scan all sensors in array (buttons and sliders)
   CSD_UpdateAllBaselines(); //Update all baseline levels;

   //detect if any sensor is pressed
   if(CSD_bIsAnySensorActive()) {
       // Add user code here to proceed the sensor touching

   }

   // now we are ready to send all status variables to chart program
   // communication here
        // Get the values from CSD User Module:
        MBM_MyMailbox.Outbox.bSensorOnOff = CSD_baSnsOnMask[0]; // Sensor On/Off
        MBM_MyMailbox.Outbox.iRawCounts = CSD_waSnsResult[0];// Raw Counts
   MBM_MyMailbox.Outbox.iDifference = CSD_waSnsDiff[0];  // Difference
        MBM_MyMailbox.Outbox.iBaseline = CSD_waSnsBaseline[0];  // Baseline

   // Set the values to CSD User Module:
        CSD_baBtnFThreshold[0] = MBM_MyMailbox.Inbox.bBtnFThreshold; // Set Button
Finger Threshold
    }
}
```

The second C example illustrates how to use the MBM User Module to communicate with the system host through the EZI2C User Module. The MailBox Manager Wizard settings are shown in Figure 7:

Figure 7. MailBox Manager Wizard Settings for the Second Code Example



```
//----------------------------------------------------------------------------
// Sample C code for the MBM User Module
//----------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

BYTE bControl1;
int  iControl2;
BYTE bInput1;
int  iInput2;

void main(void)
{
M8C_EnableGInt;  // Enable Global Interrupts
  MBM_InitializeMailbox();  // Init MBM UM

// Init EzI2Cs UM:
  EzI2Cs_SetRamBuffer(sizeof(MBM_MyMailbox), 4, (BYTE *) &MBM_MyMailbox);
  EzI2Cs_Start();                             // Turn on I2C
  EzI2Cs_EnableInt();

// Init Outbox Variables:
  bControl1 = 2;
iControl2 = 1000;

while (1) {

// now we send all status variables to chart program:
    MBM_MyMailbox.Outbox.Outbox0 = bControl1; // Send first Outbox Variable
    MBM_MyMailbox.Outbox.Outbox1 = iControl2; // Send second Outbox Variable

    // now we receive all status variables from Host Device:
    bInput1 = MBM_MyMailbox.Inbox.Inbox0; // Get first Inbox Variable
    iInput2 = MBM_MyMailbox.Inbox.Inbox0; // Get second Inbox Variable
  }
}
```

Configuration Registers

None

Version History

Version	Originator	Description
1.0	DHA	Initial Version
1.1	DHA	Added constants to support new parts: CY8C20x34-12SXI

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.