



APPLICATION NOTE

Operating the MPSC as a UART

February 6, 2002
AN-72

1. Introduction

Marvell's GT-642xx and its GT-96100A, GT-96122, and GT-96132 devices include Multi Purpose Serial Controllers (MPSC). These MPSCs can be programmed to be used as Universal Asynchronous Receiver-Transmitter (UART) devices.

This application note outlines the basic operation of the MPSC and the differences between the MPSC as a UART and the commonly used UART devices, such as the 16552 and the 8530 UARTs. Sample code for basic UART operation can be found in the attached file. This code is for the GT-96100A, GT-96122, and GT-96132; to use it for the GT-642xx, different registers, offsets, and initializations must be used.

Flow Control implementation is also described, with the aid of a connectivity diagram (see [Figure 3 on page 5](#)) and a timing chart (see [Figure 4 on page 5](#)).

2. MPSC Description

2.1 System Initialization

The GT-642xx and the GT-96100A, GT-96122, and GT-96132 devices incorporate MPSCs that can be used for different serial communication protocols. To operate the MPSC, follow the initialization steps described in [Section 2.1.1](#) through [Section 2.1.7](#) and depicted in [Figure 1](#).

2.1.1 Internal Routing and Pin Assignment

To keep the pin count and the package of the GT-642xx and the GT-96100A, GT-96122, and GT-96132 devices as small as possible, many of the pins on the device are multiplexed, so that they can be used for various tasks. To set up a UART over an MPSC, the device pins need to be configured for this activity. Pins need to be defined as input or output, and they need to be connected to the appropriate internal unit. In the sample code, MPSC 0 is used so that specific pins are connected internally to this unit. This is done by initializing the following registers:

- MRR — The Main Routing register — defines which pins will be connected to which function (MPSC, Ethernet, General purpose, etc.). The sample code initializes this register to connect Port A to MPSC 0.
- GPIO — General Purpose Input/Output registers — for GT-96100A, GT-96122, and GT-96132 devices only, configure the direction of the multiplexed pins as input or output.
- GPC — General Purpose Configuration registers — for GT-96100A, GT-96122, and GT-96132 devices only, define whether the multiplexed pins are used as I/O pins or as general purpose pins.
- PRR — Port Routing registers — for GT-96122 and GT-96132 devices only, define whether the multiplexed pins are used as I/O pins to interface directly to MPSC or to TDM.

2.1.2 Communication Unit Initialization

For GT-96100A, GT-96122, and GT-96132 devices only, set these registers:

- SGCR — SDMA Group Configuration register.
- Initialize the GT-96100A CIU (Communication Interface Unit) Arbiter Configuration register or the GT-96122/ GT-96132 CAU (Communication Arbiter Unit) Arbiter Configuration register.

<http://www.marvell.com>

2.1.3 Memory Initialization

Prepare the descriptors and buffer space in memory. See the following functions in the sample code:

- `SetRxDesc(DESC_RX0, DATA_RX0)`: This function initializes the receive (Rx) descriptors and allocates their chains and buffers.
- `SetTxDesc(DESC_TX0, DATA_TX0)`: This function initializes the transmit (Tx) descriptors and allocates their chains and buffers.

2.1.4 SDMA Receive and Transmit Initialization

Initialize the SDMA parameters including descriptor pointers. The SDMA subunit of the GT-96100A, GT-96122, and GT-96132 have two groups of DMAs — Group0 and Group1. Each group has one DMA per MPSC channel. The sample code uses channel 0 of group 0, for receive and transmit. A write to the first descriptor allocated to the following descriptor's pointers occurs:

- `SCRDP(GROUP0,0)` — The current Rx descriptor pointer of group 0 of the SDMA and channel 0 of the MPSC
- `SCTDP(GROUP0,0)` — The current Tx descriptor pointer of group 0 of the SDMA and channel 0 of the MPSC
- `SFTDP(GROUP0,0)` — The first Tx descriptor pointer of group 0 of the SDMA and channel 0 of the MPSC.

2.1.5 Clock Initialization

The GT-642xx and the GT-96100A, GT-96122, and GT-96132 devices allow great flexibility in assigning a clock (also called a BRG — Baud Rate Generator) to control the frequency of a serial communication port. To assign a clock, the following registers need to be initialized:

- `RCRR` and `TCRR` — Receive and Transmit Clock Routing registers — route a clock source to the serial communication port. The sample code uses BRG 0, which is the default; therefore, these registers are left at their original value.



Note

Marvell recommends using a clock frequency at the BRG input that divides into one of the common UART baud rates (115,200, 9600, 2400) for example use a 3.6864 MHz oscillator.

- `BCR` — The BRG Configuration register — sets the BRG configuration, to the baud rate that you require.

2.1.6 MPSC Initialization

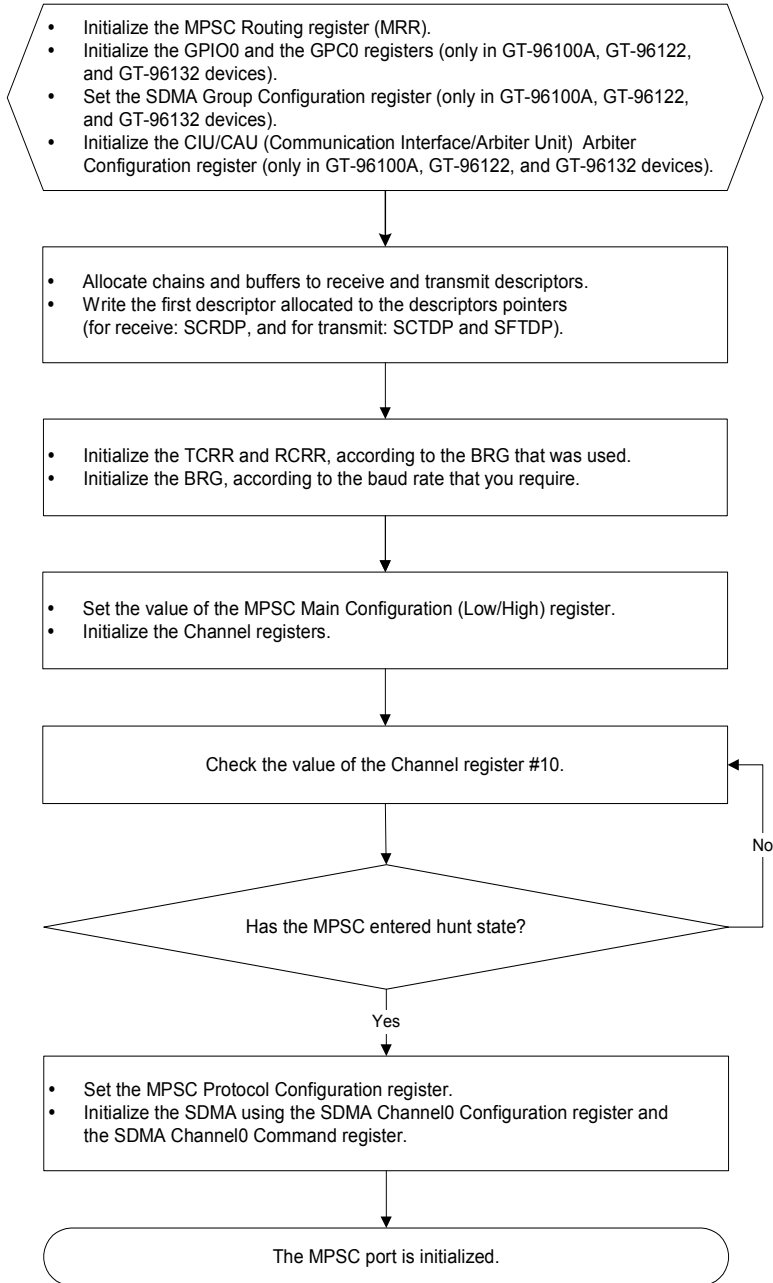
Initialize the MPSC parameters. The sample code uses MPSC 0 as a UART for the following registers:

- `MMCRL(0)` and `MMCRH(0)`: Initialize the Main Configuration (Low/High) registers of the MPSC 0.
- `CHRx`: Initialize the Channel registers for UART protocol mode.
- `CHR(10)`: Check the value of Channel register `CHR(10)`.
- `MPCR(0)`: Set the Protocol Configuration register of MPSC 0.

2.1.7 SDMA Configuration and Command Register Initialization

Initialize the SDMA Configuration register and the SDMA Command register:

- `SDC(GROUP0,0)` — SDMA Channel0 Configuration register
- `SDCM(GROUP0,0)` — SDMA Channel0 Command register.

Figure 1: MPSC Initialization Sequence

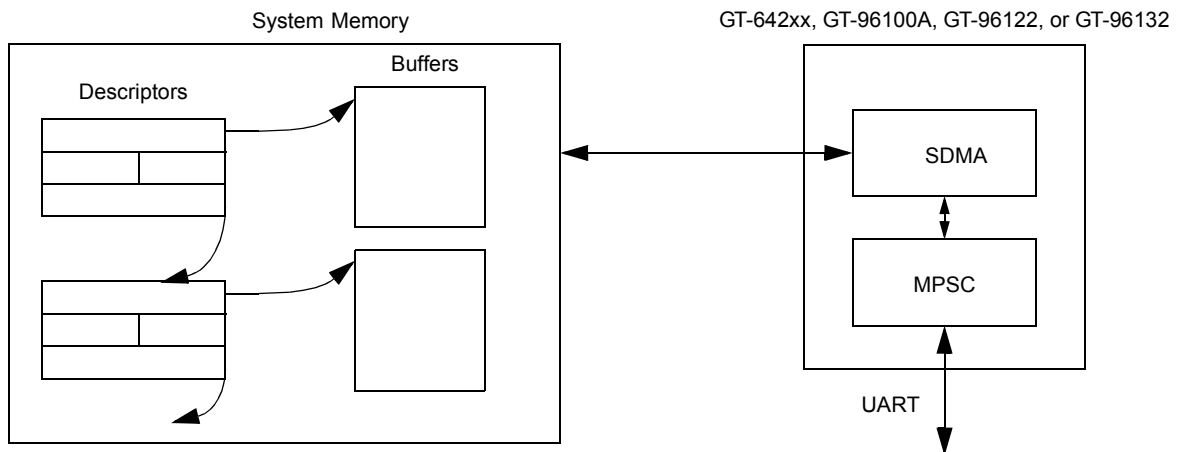
2.2 MPSC Operation

These MPSCs operate as follows:

- Transmit from the GT-642xx, GT-96100A, GT-96122, or GT-96132 device — Data is pulled from the buffer pointed to by a transmit descriptor pointer in the SDMA. This data is moved to the MPSC that transmits this data on the serial line.
- Receive by the GT-642xx, GT-96100A, GT-96122, or GT-96132 device — Data received by the MPSC is moved to the SDMA. The SDMA stores the data in the buffer pointed to by a receive descriptor pointer in the SDMA.

Figure 2 illustrates the basic functionality of these MPSCs, and shows some of the resources involved in this activity.

Figure 2: System Resources Used During MPSC Operation



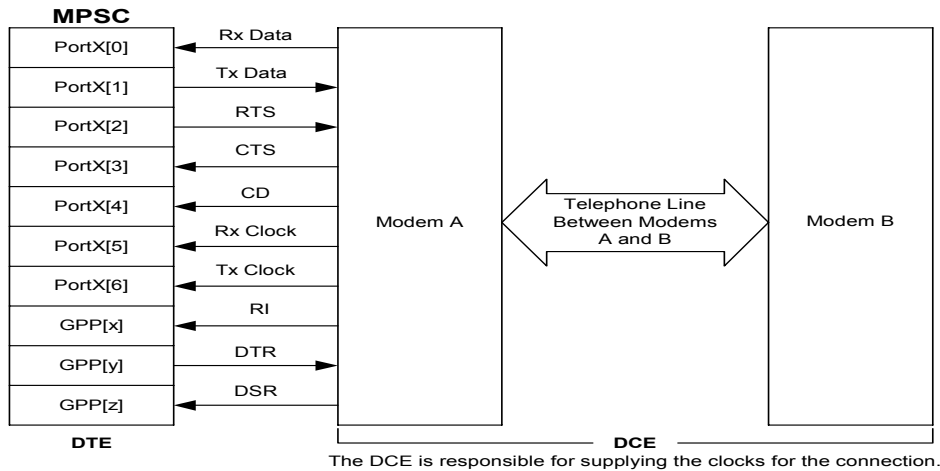
3. Flow Control

Flow control for the UART between DTE (data terminal equipment, for example, computers) and DCE (data communication equipment, for example, modems) is achieved using the RTS, CTS, CD, DTR, and DSR signals.

Before sending a character, the DTE requests permission by asserting its RTS (request to send) output. No information will be sent until the DCE grants permission by using the CTS (clear to send) line. If the DCE cannot process new requests, the CTS signal will be de-asserted. The assumption is, that the DTE always processes incoming information faster than the DCE sends it.

For further control of the information flow, both devices have the ability to signal their status to the other side. For this purpose, the DTR (data terminal ready) and DSR (data set ready) signals are present. The DTE uses the DTR signal to indicate that it is ready to accept information, and the DCE uses the DSR signal for the same purpose. These signals are in one direction only.

Figure 3 provides a connectivity diagram showing the connection between the GT-642xx, GT-96100A, GT-96122, or GT-96132 device (the DTE) and the modem (the DCE).

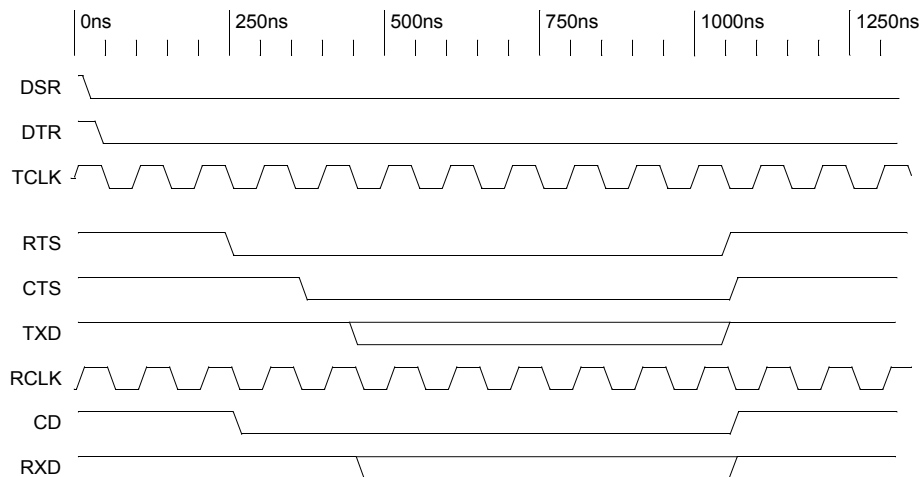
Figure 3: DTE — DCE Connectivity Diagram

RI - Ring Indicator indicates that the modem received a ring from the far end.

DSR - Data Set Ready indicates that the modem is up (configured and ready for traffic). This usually occurs after reset.

DTR - The DTE (in this case, the GT-642xx, GT-96100A, GT- 96122, or GT-96132) is up (configured and ready for traffic).

Figure 4 provides a connectivity timing chart for the DTE-DCE connection of the GT-642xx, GT-96100A, GT-96122, or GT-96132 device (the DTE) and the modem (the DCE).

Figure 4: DTE — DCE Timing Chart

4. Operation of the MPSC Versus a Dedicated UART Device

UART dedicated devices, such as the 16552 and the 8530, support the traditional poll-and-read-register mechanism, meaning that any incoming data is stored in an on-chip register and the data to be transmitted is taken from an on-chip register. The system's interface with the chip is through these registers. As described in the previous section, the MPSC operates differently. The SDMA will move data received from the MPSC to the memory (SDRAM), and will move data to be transmitted from the memory to the MPSC.

Although this implementation of using system memory results in higher efficiency, it might cause a problem for systems that want to implement a debug port. A debug port is perceived as a part of the system that can be as independent as possible, and should not, usually, require the use of system memory (see [Section 5](#)).

5. Operating the MPSC as a Debug Port

5.1 Initialization

The Baud Rate Generator should be set correctly to the HyperTerminal baud rate as explained in [Section 2.1.5 Clock Initialization](#).

5.2 UART Transmit (Tx): Using TCS to Generate Messages

To transmit data when the UART is in Debug Port mode, the system can use the TCS mechanism for transmitting data to the port before the SDRAM is accessible. The system can generate messages by loading the data to be transmitted to the TCS field in the MPSC CHR1 register and then forcing a transmission of this data by issuing a TCS command in the MPSC CHR2 register.

The GPC, GPIO, MRR, and PRR registers are configured in the same manner as when operating the UART in regular mode.

5.3 UART Receive (Rx): Using CFR to Capture Messages



Note

The GT-96100A-B-1 and later versions, GT-96122-B-x, and GT-96132-B-x support this feature.

To receive data when the UART is in Debug Port mode, the system can use the UART in Debug Port mode enable bit in the MPSC CFR (CHR4) register (see the note below). Setting this bit, together with a value of 0x0 in the CFR field of the same register, enables the UART as Debug Port mode. In this mode the MPSC receives and stores the data at the RCRn field in the MPSC CHR10 register. The data is received/stored according to the Control Character setting of CTL1 field of the MPSC CHR5 register. Setting the Valid bit enables storage of received data. Setting the Interrupt bit enables maskable interrupt generation on every byte received. In this case, bit 6 of the relevant MPSC Mask register, offset: 0x103aa0, 0x103aa4..0x103abc, should be set to 1 (see [Table 1](#)). After reading the received data in the RCRn field, the field should be cleared by writing 1 to all the set bits. Also, if an interrupt was generated, bit 6 of the relevant MPSC Cause register, offset: 0x103a20, 0x103a24..0x103a3c, should be cleared by writing 0 to this bit (see [Table 1](#)).

**Notes**

- Bit 29 (Z) of the Control Filter register (CFR) is the UART enable bit for Debug Port mode. Setting this bit has meaning only when the CFR field is set to 0x0.
- For Bit 12 (INT) of the UART Control Character register format, 1 = Generate interrupt upon receiving this character. In UART as Debug Port mode, an interrupt is generated on every character received.
- For Bit 15 (V) of the UART Control Character register format, 1 = Entry is valid. In UART Debug Port mode, this setting will enable storing the character received to the RCRn field.
- Bits 23:16 (RCRn) of the UART Event Status register (ESR) contain the received byte.

**Table 1: MPSC0 Cause Register, Offset: 0x103A20 and
MPSC0 Mask Register, Offset: 0x103AA0**

Bits	Field Name	Function	Initial Value
6	Mpsc0RFSC/ Mpsc0RCC	MPSC0 Rx Flag Status Change (HDLC mode) MPSC0 Received Control Character (Bisync, UART modes)	0

5.4 Sample Code for MPSC0

For GT-642xxA define the these register offsets as follows:

```
#define BRG0_CONFIGURATION_REGISTER      0xB200
#define UART_CONFIGURATION_REGISTER_VALUE 0x10000
#define MPSC0_CAUSE                      0xB804
#define MPSC0_MASK                       0xB884
#define MPSC0_MAIN_CONFIGURATION_LOW     0x8000
#define MPSC0_MAIN_CONFIGURATION_HIGH   0x8004
#define MPSC0_PROTOCOL_CONFIGURATION     0x8008
#define CHANNEL0_REGISTER1              0x800C
#define CHANNEL0_REGISTER2              0x8010
#define CHANNEL0_REGISTER3              0x8014
#define CHANNEL0_REGISTER4              0x8018
#define CHANNEL0_REGISTER5              0x801C
#define CHANNEL0_REGISTER6              0x8020
#define CHANNEL0_REGISTER7              0x8024
#define CHANNEL0_REGISTER8              0x8028
#define CHANNEL0_REGISTER9              0x802C
#define CHANNEL0_REGISTER10             0x8030
#define CHANNEL0_REGISTER11             0x8034
#define MPSC_ROUTING_REGISTER           0xB400
#define MPSC_ROUTING_REGISTER_VALUE     0x7FFE38
```

For GT-GT-96100A, GT-96122, and GT-96132 define the these register offsets as follows:

```
#define BRG0_CONFIGURATION_REGISTER      0x102A00
#define UART_CONFIGURATION_REGISTER_VALUE 0x50000
#define MPSC0_CAUSE                      0x103A20
#define MPSC0_MASK                       0x103AA0
#define MPSC0_MAIN_CONFIGURATION_LOW     0x0A00
#define MPSC0_MAIN_CONFIGURATION_HIGH    0x0A04
#define MPSC0_PROTOCOL_CONFIGURATION     0x0A08
#define CHANNEL0_REGISTER1               0x0A0C
#define CHANNEL0_REGISTER2               0x0A10
#define CHANNEL0_REGISTER3               0x0A14
#define CHANNEL0_REGISTER4               0x0A18
#define CHANNEL0_REGISTER5               0x0A1C
#define CHANNEL0_REGISTER6               0x0A20
#define CHANNEL0_REGISTER7               0x0A24
#define CHANNEL0_REGISTER8               0x0A28
#define CHANNEL0_REGISTER9               0x0A2C
#define CHANNEL0_REGISTER10              0x0A30
#define CHANNEL0_REGISTER11              0x0A34
#define MPSC_ROUTING_REGISTER             0x101A00
#define MPSC_ROUTING_REGISTER_VALUE       0x00FC0000
#define CAU_ARBTR_CONFIG_REG              0x101AC0
#define CAU_ARBTR_CONFIG_REG_VALUE        0x800003FE
#define GPC0_REG                          0x101A00
#define GPC0_REG_VALUE                     0xFFFF0000
#define GPIO0_REG                         0x101A20
#define GPIO0_REG_VALUE                     0x46460000
```

For GT-642xxA, GT-96100A, GT-96122, and GT-96132:

```
// declare all variables and set their values to 0.
```

```
// set the UART control characters in char5 :
// CHAR - is 8f ,
// Reserved - 0
// INT - 1
// CO - 0
// R - 0
// V - 1
```

```
SetUartCtl(0,"8F",0,1,0,0,1);
```

```
MPSC[0].BCE = 0x0; /* CFR = 0x00 */
```



```
/* Set the MPSC main routing register , the port should be connected to MPSC */
Write (MPSC_ROUTING_REGISTER,MPSC_ROUTING_REGISTER_VALUE);

/* For GT-96100A, GT-96122, and GT-96132:
Configure GPIO, GPC and CIU (GT-96100A) or CAU (GT-96122 and GT-96132) communication arbiter,
as explained in Section 2.1 System Initialization. */
Write (CAU_ARBTR_CONFIG_REG,CAU_ARBTR_CONFIG_REG_VALUE);
/* for UART over MPSC# 0 or 1 */
Write (GPC0_REG,GPC0_REG_VALUE);
Write (GPIO0_REG,GPIO0_REG_VALUE);

MMCRL.MODE = 0x4; /* UART Mode */
MMCRL.ER = 1; /* enable receive */
MMCRL.ET = 1; /* enable transmit */
MMCRH.RCDV = 1; /* x8 clock mode */

// set the BRG to baud rate of 115200 , ClkIn is BclkIn (3686400 Hertz).
// CDV is equal zero (ClkIn = ClkOut).
// TCDV and RCDV configured to X32.
// (3686400 / 32 = 115200.
Write(BRG0_CONFIGURATION_REGISTER,UART_CONFIGURATION_REGISTER_VALUE);

Write(MPSC0_MAIN_CONFIGURATION_LOW,MMCRL);
Write(MPSC0_MAIN_CONFIGURATION_HIGH,MMCRH); /* activate mpsc */

MPCR.ISO = 0; /* asynchronous */
MPCR.SBL = 0; /* 0; */ /* one stop bit */
MPCR.CL = 3; /* number of data bits 8 */
Write(MPSC0_PROTOCOL_CONFIGURATION,MPCR); /* config uart */

CHR4.CFR = 0; /* MPSC as UART switch chr4[7:0](cfr)=0 */
CHR4.Z = 1; /* MPSC as UART switch chr4[29](z)=1 */
Write(CHANNEL0_REGISTER4,CHR4); /* start MPSC as UART mode */

CHR5.CTL[0].V = 1; /* enables storing of the received data */
CHR5.CTL[0].INT = 1; /* enable maskable interrupt */
Write(CHANNEL0_REGISTER5,CHR5); /* config MPSC as UART */

MPCS0MR.RCC = 1 /* enable bit 6 of MPSC Interrupt Cause register */
Write(MPSC0_MASK,MPCS0MR); /* enable MPSC as UART byte recieved interrupt */

CHR2.EH = 1; /* enter hunt command */
```

```

Write(CHANNEL0_REGISTER2,CHR2); /* start hunting */

Byte = 0x55;

for (i=0; i<1; ) { /* loop forever */

    CHR1.TCS = Byte; /* set byte to transmit */
    Write(CHANNEL0_REGISTER1,CHR1); /* config MPSC as UART transmit byte */

    CHR2.TCS = 1; /* MPSC as UART transmit command */
    Write(CHANNEL0_REGISTER2,CHR2); /* start MPSC as UART transmit */

    Read(CHANNEL0_REGISTER2,&CHR2); /* poll tcs command */
    while (CHR2 & 0x00000200) { /* check TCS bit */
        Read(CHANNEL0_REGISTER2,&CHR2);
    }

    Read(MPSC0_CAUSE,&MPSC0CR); /* poll interrupt bit */
    while (MPSC0CR & 0x00000040 == 0) { /* check RCC interrupt bit */
        Read(MPSC0_CAUSE,&MPSC0CR);
    }

    Read(CHANNEL0_REGISTER10,&CHR10); /* read recieved value */
    Byte = (CHR10 >> 16) & 0xff; /* extract byte value */

    Write(CHANNEL0_REGISTER10,CHR10); /* clear RCRn field in the CHR10 register*/
    Write (MPSC0_CAUSE,MPSC0CR & 0xfffffbfbf); /* clear interrupt bit */

} /* end loop */

```

6. Revision History

[Table 2](#) provides the revision history of this application note.

Table 2: Document History

Document Type	Revision	Date	Comments
Application Note	1.0	May 31, 2001	
Application Note	-	July 11,2001	Revision Update (first Marvell release of this AN)

Table 2: Document History (Continued)

Document Type	Revision	Date	Comments
<ol style="list-style-type: none"> Throughout this document, "GT-961xx" was changed to "GT-96100A, GT-96122, and GT-96132". Figure 1 on page 3 was revised. The sample code in Section 5.4 Sample Code for MPSC0 on page 7 was changed to include code that may be used for GT-642xxA, GT-96100A, GT-96122, and GT-96132 devices. A document control number has been added by corporate document control. The revision number has been changed to Rev. - by corporate document control. 			
Application Note	A	August 30, 2001	Revision Update
<ol style="list-style-type: none"> Section 5.1 Initialization was added. In Section 5.4 Sample Code for MPSC0, the following was added to the code "For GT-642xxA define the these register offsets as follows:" <pre>#define BRG0_CONFIGURATION_REGISTER 0xb200</pre> In Section 5.4 Sample Code for MPSC0, the following was added to the code "For GT-GT-96100A, GT-96122, and GT-96132 define the these register offsets as follows:" <pre>#define BRG0_CONFIGURATION_REGISTER 0x102A00</pre> In Section 5.4 Sample Code for MPSC0, the following was added to the code "For GT-642xxA, GT-96100A, GT-96122, and GT-96132:" <pre>// set the BRG to baud rate of 115200 , ClkIn is BclkIn (3686400 Hertz). // CDV is equal zero (ClkIn = ClkOut). // TCDV and RCDV configured to X32. // (3686400 / 32 = 115200. Write(BRG0_CONFIGURATION_REGISTER,0x00050000);</pre> 			
Application Note	B	December 19, 2001	Revision Update
<ol style="list-style-type: none"> Added a sentence about the connectivity diagram and the timing chart to Section 1. Introduction. In Section 2.1.1 Internal Routing and Pin Assignment, added PRR to the list of registers. In Section 2.1.2 Communication Unit Initialization, added reference to CIU for the GT-96100A and the CAU for the GT96122 and the GT-96132. In Section 2.1.5 Clock Initialization, added a note concerning the clock frequency. In Figure 1: MPSC Initialization Sequence, added CAU to the explanation in the last bullet of the first block. Added Section 3. Flow Control. In Section 5.2 UART Transmit (Tx): Using TCS to Generate Messages, explained that the configuration of the GPC, GPIO, MRR, and PRR registers does not change when operating the MPSC as a debug port. 			
Application Note	C	February 6, 2002	Revision Update
<ol style="list-style-type: none"> In Section 5.4 Sample Code for MPSC0, added "<code>#define UART_CONFIGURATION_REGISTER_VALUE 0x10000</code>", "<code>#define MPSC_ROUTING_REGISTER 0xB400</code>" and "<code>#define MPSC_ROUTING_REGISTER_VALUE 0x7FFE38</code>" to the defines for the GT-642xx and added "<code>#define UART_CONFIGURATION_REGISTER_VALUE 0x50000</code>" and from <code>#define MPSC_ROUTING_REGISTER0x101A00</code> to the end of the list to the defines for the GT-96100A, GT-96122, and GT-96132. After the code line <code>MPSC[0].BCE = 0x0; /* CFR = 0x00 */</code>, nine lines were added to the code up to and including the line <code>Write (GPIO0_REG,GPIO0_REG_VALUE);</code>. The code line <code>Write(BRG0_CONFIGURATION_REGISTER,UART_CONFIGURATION_REGISTER_VALUE);</code> was changed to replace the register offset with a define value. Changed the value after the equal sign in the code line <code>MPCR.SBL = 0; /* 0; */ /* one stop bit */</code>. Changed the code line <code>while (CHR2 & 0x00000200) { /* check TCS bit */</code>. 			



Preliminary or Advanced Information: This document provides preliminary or advanced information about the product described. All specifications described herein are based on design goals only. **Do not use for final design.** Visit Marvell's web site at www.marvell.com or call 1-866-674-7253 for the latest information on Marvell products.

DISCLAIMER

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Marvell. Marvell retains the right to make changes to this document at any time, without notice. Marvell makes no warranty of any kind, expressed or implied, with regard to any information contained in this document, including, but not limited to, the implied warranties of merchantability or fitness for any particular purpose. Further, Marvell does not warrant the accuracy or completeness of the information, text, graphics, or other items contained within this document. Marvell makes no commitment either to update or to keep current the information contained in this document. Marvell products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use Marvell products in these types of equipment or applications. The user should contact Marvell to obtain the latest specifications before finalizing a product design.

Marvell assumes no responsibility, either for use of this product or for any infringements of patents and trademarks, or other rights of third parties resulting from its use. No license is granted under any patents, patent rights, or trademarks of Marvell. This product may include one or more optional functions. The user has the choice of implementing any particular optional function. Should the user choose to implement any of these optional functions, it is possible that the use could be subject to third party intellectual property rights. Marvell recommends that the user investigate whether third party intellectual property rights are relevant to the intended use of this product and obtain licenses as appropriate under relevant intellectual property rights.

Marvell comprises Marvell Technology Group Ltd. (MTGL) and its subsidiaries, Marvell International Ltd. (MIL), Marvell Semiconductor, Inc. (MSI), Marvell Asia Pte Ltd. (MAPL), Marvell Japan K.K. (MJKK), Galileo Technology Ltd. (GTL) and Galileo Technology Inc. (GTI).

Copyright © 2002 Marvell. All Rights Reserved. Marvell, GalNet, Galileo, Galileo Technology, Fastwriter, Prestera, Moving Forward Faster, Alaska, the M logo, GalTis, GalStack, GalRack, NetGX, the Max logo, Communications Systems on Silicon, and Max bandwidth trademarks are the property of Marvell. All other trademarks are the property of their respective owners.

Marvell Semiconductor, Inc.
2350 Zanker Road
San Jose, CA 95131
Phone: (408) 367-1400, Fax: (408) 367-1401

E-mail: commsales@marvell.com