# The MC68302 provides peripheral and communications functions for PowerPC$^{TM}$ Microprocessors. [*]

By:- Gordon Lawton, Data Communications Applications, East Kilbride.
Colin MacDonald, RISC Applications, East Kilbride.

## Introduction

The MC68302 integrated multiprotocol processor (IMP) can be used to provide a PowerPC microprocessor based system with many useful communications and systems functions. A key feature of the IMP is that it supports a mode in which it's internal CPU is disabled, allowing the device to act as a highly integrated peripheral with on-chip RAM. By using this internal RAM (instead of external RAM) to buffer communications transfers, the MC68302 does not have to become an external bus master. As a result, interface logic can be made less complex, allowing it to be easily integrated into a wide variety of systems. This document describes the design of a simple, low-cost interface between a PowerPC 60x microprocessor and the IMP. It also provides an overview of the features of the IMP and discusses how these can be used in a PowerPC microprocessor based system.

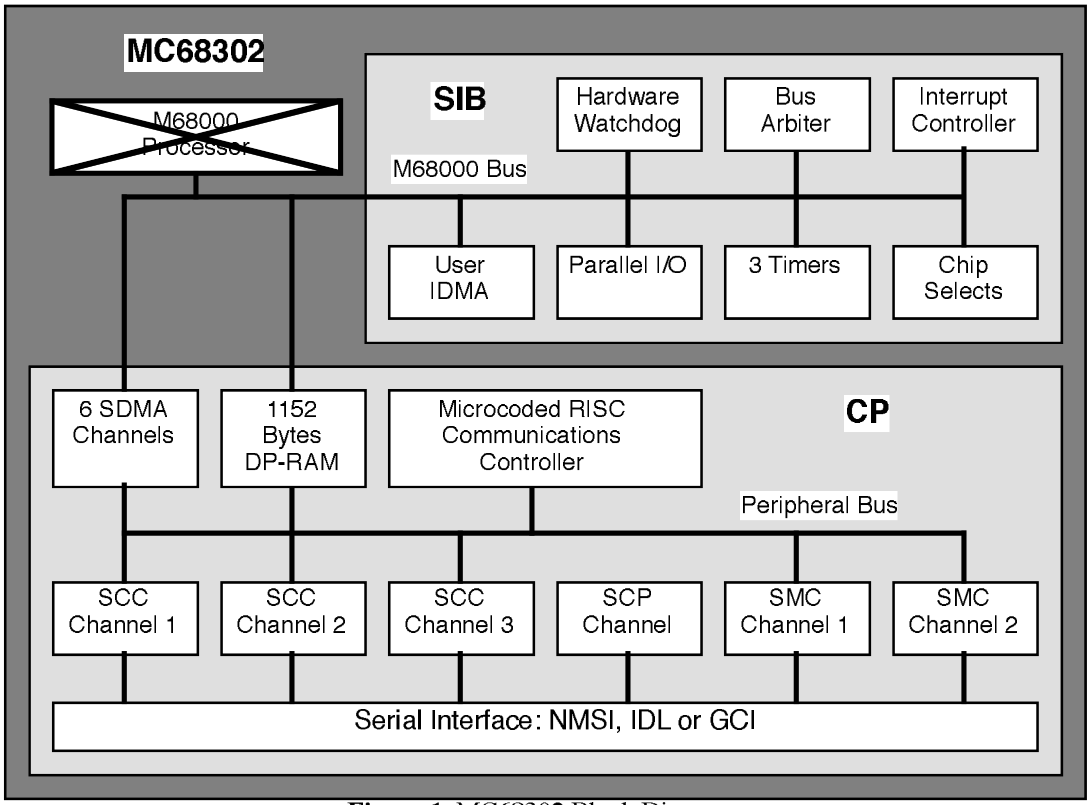

**Figure 1** MC68302 Block Diagram

## MC68302 Overview

[*]. PowerPC$^{TM}$, PowerPC 601$^{TM}$, PowerPC 603$^{TM}$ and PowerPC604$^{TM}$ are trademarks of International Business Machines Corporation and are used by Motorola, Incorporated, under license from International Business Machines Corporation. All other trademarks are recognised.

The MC68302 Integrated Multiprotocol Processor is normally controlled by an on chip M68000 CPU. However it can be configured to operate as a peripheral to an external processor such as the Power PC 60X. All of the MC68302 peripherals contained in the System Integration Block (SIB) and the Communications Processor (CP) are available to the external host in this mode, as illustrated in figure 1. The dual-port RAM offers 576 bytes of additional system RAM for general use, such as data buffer storage, and 576 bytes of parameter RAM used to configure and control the CP and SIB.

The CP consists of a RISC processor, three SCCs (serial communications controllers), two serial DMA channels for each SCC, a serial communications port (SCP), two serial management controllers (SMCs) and a programmable serial interface,. The RISC processor is a separate processor from the 68000 core, and is dedicated to the service of the SCCs, SCP and SMC, therefore, in Disable-CPU mode all the communications functionality is available. It uses these channels to implement the user-selected protocols, and to manage the six serial DMA channels that transfer data between the SCCs and the memory. Data from each SCC may be transferred into, or out from 8 buffers. This operation requires no intervention from the external processor thanks to serial DMA. The internal bus structure of the MC68302 enables the SDMA to transfer data to the internal Dual Port RAM over the peripheral bus. Therefore, if the serial buffers are located in the DP RAM, the MC68302 does not required bus ownership when transmitting and receiving data. This feature makes the MC68302 particularly attractive as a serial peripheral, as no arbitration logic is required and there is no impact to the Power PC bandwidth from the serial DMAs. When the external processor requires data it can read directly from the internal Dual Port RAM.

The three built-in serial channels support high data rates and offer a wide selection of standard communications protocols including; HDLC, UART, BISYNC, DDCMP or Fully Transparent Operation. The physical interface support standard NMSI (Non-Multiplexed Serial Interface) as well as The multiplexed modes include IDL, GCI, (both used in ISDN applications) and a PCM Highway. The MC68302 provides a Power PC system with serial capability ranging from 9,600 baud UART channels to 2Mb/s X.25 links.

The SIB incorporates general-purpose peripherals that eliminate much glue logic. It includes an independent DMA controller (IDMA), an interrupt controller, parallel I/O ports, an 1152-byte dual-port RAM, two timers and a watchdog timer. The SIB also has chip-select lines, wait-state-generation logic, a bus arbiter, an on-chip clock generator, and a hardware watchdog. In a Power PC system it is unlikely the IDMA or the chip selects provided on the MC68302 would be used as they are design for a M68000 system and thus lower performance, but the remaining SIB peripherals are extremely useful in such a system.

A total of 28 Parallel I/O lines are multiplexed with the three SCCs and SCP. Regardless of the way the on-chip peripherals are configured, at least five I/O lines are always available, four of which can interrupt the external host. The RISC Communications Controller also generates interrupts to the on-chip interrupt controller. Interrupts can be presented to the Power PC processor on a single interrupt level which can determine the source by simply reading the Interrupt In-Service Register on the MC68302.

Two general-purpose 16-bit timers support capture and output pulse/toggle options. The Software Watchdog Timer includes a dedicated output pin that may be used to reset or interrupt the Power PC host. Finally, the hardware watchdog can terminate bus cycles if a illegal read or write cycle is made to the MC68302 by asserting the *BERR pin.

## POWERPC60X to MC68302 INTERFACE

The PowerPC 601, PowerPC 603 and PowerPC 604 microprocessors are all members of the PowerPC 60x family. Although not identical, the system interfaces of these devices are very similar, allowing the design of generic PowerPC60x systems. The PowerPC60x system interface is a synchronous, pipelined interface that supports a wide variety of transaction types including pipelined and split-bus transactions. Of course, not all transaction types make sense for all systems. The fewer types supported, the simpler the system control logic can be made.The PowerPC60x to IMP interface described below is designed for single processor, non-pipelined system, in which the processor is parked on the bus and it's data bus grant line is tied low.

The simple interface described here (see Figure 2) uses a single PLD22V10 and a 16-bit non-inverting transceiver.It can also be used in low power systems as 3.3V versions of all components are available.

In PowerPC603 and PowerPC604 based systems, the control PLD (IMPI.PLD) is clocked by either SYSCLK or it's equivalent. The clocking scheme for the PowerPC601 microprocessor is a little different: in this case BCLK_EN should be held low and the IMPI.PLD clocked by PCLK_EN or it's equivalent. In both cases, the PLD is effectively clocked by the PowerPC60x bus clock, so it can function over a wide range of PowerPC60x bus clock frequencies. Since the design performs asynchronous MC68302 internal accesses, the design will also function at different IMP's clock rates. Determining the operational frequency range of the design for a chosen PLD speed and IMP frequency is left to the reader, however, with 7.5nS PLDs and a 16MHz IMP, the maximum frequency of the interface exceeds 66MHz. At this speed accesses to the MC68302 take approximately 16 bus clocks.

In terms of data bus widths, PowerPC60x processors have a 64-bit data buses whereas the MC68302 has a 16 bit data bus, 16-bit wide internal RAM and 8-bit or 16-bit wide internal registers. Since some of the IMPs internal 8-bit registers lie next to reserved fields, the interface was designed to support byte-sized granularity. PowerPC60x devices indicate operand size (from one to eight bytes) via TSIZ(0:2) pins. The interface PLD uses TSIZ(1:2) and low-order address lines A(29:31) to control the assertion of the MC68302's data strobes UDS and LDS which are inputs when the IMP is not a bus master. In order to simplify the logic, it is assumed that only byte or half word (16 bit) accesses are made to the MC68302.

.

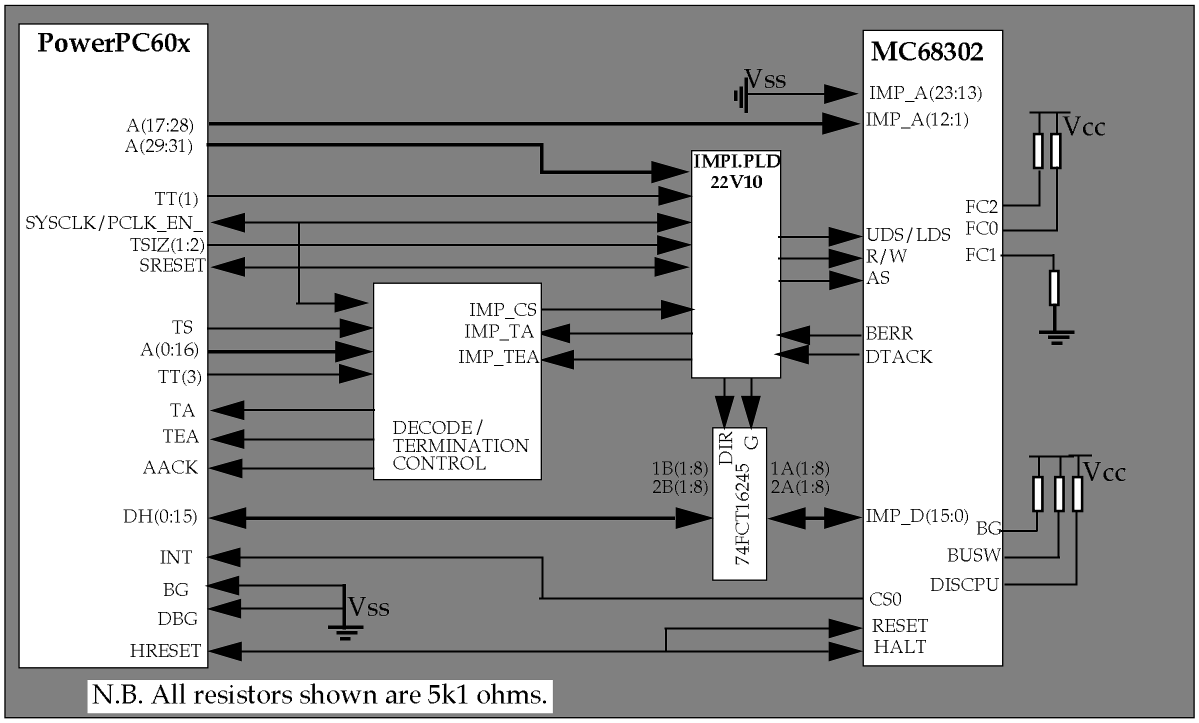

**Figure 2** PowerPC60x to MC68302 Interface Block Diagram

One aspect of the design that could be a little confusing is the mapping of PowerPC60x address and data lines to those of the MC68302. Although both of these processors employ Big Endian byte ordering (where the address of the datum is the address of it's most significant byte), they use different bit ordering conventions for their address and data buses. PowerPC60x microprocessors use Big Endian nomenclature, where the most significant position is labelled 'bit zero'. The MC68302 bit ordering, on the other hand, looks Little-Endian, where the least significant position is labelled 'bit zero'. In order to connect these buses sensibly the relative significance of connecting lines must agree. This is achieved for the two data buses by connecting in sequence, PowerPC60x lines DH(0:15) to **IMP_D(15:0).

**. In this application note the MC68302 address bus and data buses are called IMP_A(23:1) and IMP_D(15:0) respectively. This is done to distinguish them from the PowerPC60x address and data buses.

All MC68302 internal resources (including RAM and registers) are held in a 4k-byte relocatable block, requiring eleven address lines IMP_A(11:1) to span it. Giving this block a non-zero base address means at least one other address line IMP_A(12) is required. The remaining MC68302 address lines IMP_A(23:13) are tied low. This reduces the number of connections to the PowerPC60x address bus but still allows access to the IMP's system configuration registers. Due to the different widths of the processors' data buses and the design of the interface, the PowerPC60x's three least significant address lines A(29:31) are not connected to any IMP address inputs. Instead, the least significant address line on the MC68302, IMP_A(1), is connected to the forth least significant, A(28), on the PowerPC60x. This mapping is continued so that IMP_A(12:1) are connected, in sequence, to A(17:28) - as shown in Figure 2. With these address and data bus mappings, the addresses of internal IMP resources can be calculated as follows:-

Base address of MC68302 relative to PowerPC60x processor = A;
Address of MC68302 internal resource relative to the MC68302 = B;
Address of MC68302 internal resource relative to the PowerPC60x processor = C;

For 16-bit aligned MC68302 locations (8 or 16-bits wide):-
C =A+(B*4);

For 16-bit mis-aligned MC68302 locations (8-bits wide):-
C = (A+((B-1)*4)+1);

Listing 2 is a 'C' language program, with header file, that shows how to initialise an MC68302 to provide two UART channels. These channels use internal RAM to buffer communications. The values given in the program and header files assume the MC68302 is located at address 0x80000000 and that the buses are mapped as previously described.

As can be seen from the block diagram (Figure 2) and PLD equations (Listing1), the interface is designed to work with other control logic. This control logic is responsible for decoding and generating an IMP chip select and, during IMP accesses, it drives the PowerPC60x transfer termination signals (AACK, TA and TEA) in response to the IMP_TA and IMP_TEA signals from interface PLD. In fact, the design of the IMPI.PLD assumes TA and TEA will be asserted one clock after IMP_TA and IMP_TEA.

Since, in this design, the IMP is running asynchronously to the PowerPC microprocessor, IMPU_TA is asserted only after the IMP indicates (by asserting DTACK) that either data has been latched on a write cycle or that data is valid on a read cycle (see state machine - Figure3). The IMP's hardware watchdog allows accesses to unimplemented areas within the MC68302's address range to cause an assertion of the IMP's BERR signal. This in turn causes the interface PLD to assert IMP_TEA.

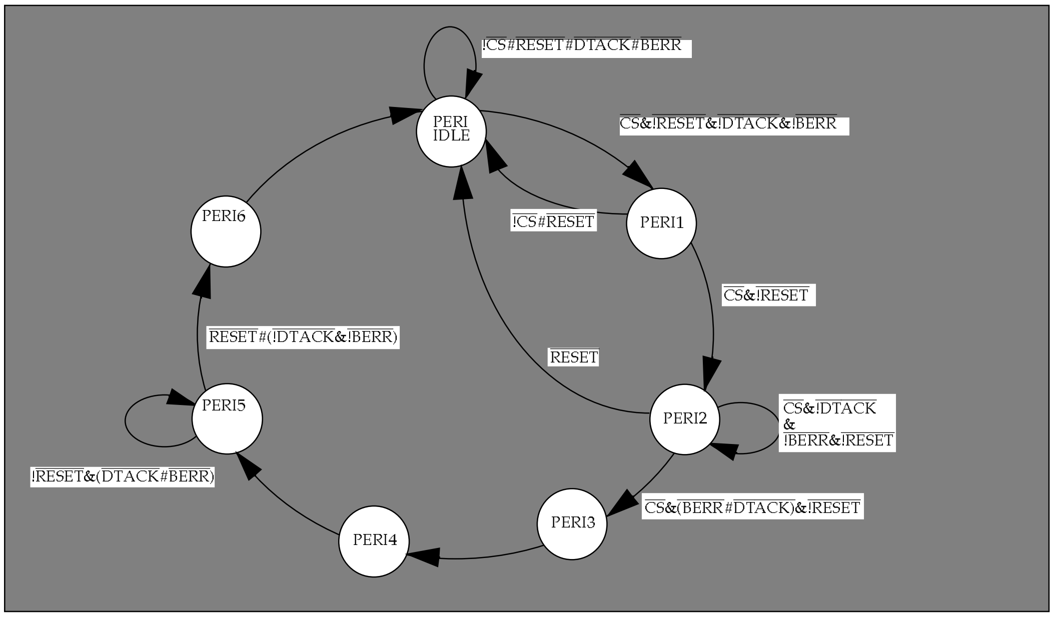

**Figure 3** IMPI.PLD State Machine

One of the useful system functions the IMP can provide in Disable CPU mode is that of an interrupt controller. Although the IMP supports seven levels of interrupt, all internal interrupt sources and four port lines (PortB 11, 10, 9 and 8) generate level 4 interrupts. An interrupt mask register allows the propagation of these interrupts to be enabled or disabled and in Disable CPU mode, the IMP can be configured so that it's CS0/IOUT2 is driven low if an unmasked level4 interrupt occurs. This line can be used drive the PowerPC60x's INT input directly.

A small number of pull-up and pull-down resistors are required to configure the MC68302 specifically for this interface. In order to access all internal registers, including the supervisor-access only Base Address Register, the IMP's function code pins FC(0:2) are set to a value of 0x5. To disable the CPU of the IMP and set it's bus width to 16-bits, DISCPU and BUSW are pulled high. In Disable CPU mode, BG becomes an input, and this is also tied high to prevent the IMP becoming a bus master.

Note that Figure 2 does not show general support logic, as this is not specific to the interface design. However, it's worth mentioning that in this design HALT, BERR, DTACK and RESET all have 5k1 pull up resistors and the MC68302 is clocked using a 16.0MHz TTL-level clock.

**Listing 1 IMPI.PLD**

```
Name            impi.pld
Partno          P00001;
Date            05/09/93;
Revision        01;
Designer        CMcD (EKB);
Company         Motorola Ltd. Copyright (C);
Assembly        Demo;
Location        ICxx;
Device          p22v10;
```

```
Format            j;

/************************************************************************ */
/* DESCRIPTION:   Control Pal for PowerPC60x to MC68302 interface      */
/* HISTORY:       Rev1.0   Production                                   */
/********************************8888*********************************/
/* Allowable Target Device Types:  22V10 10ns or faster                 */
/************************************************************************/

/**  Inputs  **/

Pin 1        =  clk              ; /*  system clock             */
Pin 2        =  sz2              ; /*  size encoding            */
Pin 3        =  sz1              ; /*  size encoding            */
Pin 4        =  s_a29            ; /*                           */
Pin 5        = !pd               ; /*  peripheral dtack         */
Pin 6        =  s_tt1            ; /*  read/write               */
Pin 7        =  s_a30            ; /*                           */
Pin 8        = !be               ; /*  berr from 302            */
Pin 9        = !ag               ; /*  access grant             */
Pin 10       = !p_cs             ; /*  peripheral chip select   */
Pin 11       = !rs               ; /*  reset from MPC60X        */
Pin 13       =                   ; /*  low order address lines  */

/** Outputs **/

Pin 14       = sty               ; /*  state variable           */
Pin 15       = !imp_tea          ; /*  imp_tea for MPC601 only  */
Pin 16       = !ena              ; /*  data buffer enable       */
Pin 17       = !imp_ta           ; /*  early ta                 */
Pin 18       = !uds              ; /*  uds for 68302            */
Pin 19       = !lds              ; /*  lds for 68302            */
Pin 20       =  rw               ; /*  R/W signal for peripheral  */
Pin 21       = !as               ; /*  AS for peripheral        */
Pin 22       = stz               ; /*  State variable0          */
Pin 23       = stx               ; /*  State variable1          */

/** Declarations and Intermediate Variable Definitions **/
rd           =  s_tt1;
wr           = !s_tt1;
wrd          = !sz2;

/* dir = 0, default to write cycles, A->B, ppc on bus A */

Field peri_sm = [stx,sty,stz];
$define PERI_IDLE       'b'000
$define PERI_1          'b'001
$define PERI_2          'b'010
$define PERI_3          'b'011
$define PERI_4          'b'100
$define PERI_5          'b'101
$define PERI_6          'b'110
$define PERI_END        'b'111

pa = p_cs ;
odd_byte = sz2 & s_a31;
evn_byte = sz2 & !s_a31;

/**  Logic Equations  **/
sequence peri_sm {
present PERI_IDLE         if !pa #  rs #  pd #  be            next PERI_IDLE;
                          if  pa & !rs & !pd & !be            next PERI_1;
present PERI_1            if  pa                              next PERI_2;
                          if !pa # rs                         next PERI_IDLE;
present PERI_2            if pa & !pd & !be & !rs             next PERI_2;
                          if pa & (be # pd) & !rs             next PERI_3;
                          if rs                               next PERI_IDLE;
present PERI_3                                                next PERI_4;
present PERI_4                                                next PERI_5;
present PERI_5            if !rs & (pd # be)                  next PERI_5;
                          if  rs # !pd & !be                  next PERI_6;
present PERI_6                                                next PERI_IDLE;
}

uds.d =  peri_sm:PERI_2 & wrd
       #  peri_sm:PERI_2 & evn_byte
       #  peri_sm:PERI_3 & wrd
       #  peri_sm:PERI_3 & evn_byte
       #  peri_sm:PERI_4 & wrd
       #  peri_sm:PERI_4 & evn_byte;

lds.d =  peri_sm:PERI_2 & wrd
      #  peri_sm:PERI_2 & odd_byte
      #  peri_sm:PERI_3 & wrd
      #  peri_sm:PERI_3 & odd_byte
      #  peri_sm:PERI_4 & wrd
      #  peri_sm:PERI_4 & odd_byte;

as.d  =  peri_sm:PERI_1
      #  peri_sm:PERI_2
      #  peri_sm:PERI_3
      #  peri_sm:PERI_4;

rw.d  = s_tt1;
imp_ta.d  = peri_sm:PERI_3;
imp_tea.d = peri_sm:PERI_3 & be;            /* imp_tea co-incident with s_ta */

/* Enables  */
imp_ta.oe = 'b'1;     /* early ta */
stx.oe    = 'b'1;     /* For simulation only */
sty.oe    = 'b'1;     /* For simulation only */
stz.oe    = 'b'1;     /* For simulation only */
ena.oe    = 'b'1;     /* bi-dir buffer and peri */
uds.oe    = 'b'1;     /* uds for peri */
lds.oe    = 'b'1;     /* lds for peri */
as.oe     = 'b'1;     /* as for peri */
rw.oe     = 'b'1;     /* rw for buf an8d peri */
imp_tea.oe= ‘b’0;     /* disable imp_tea as 601 diff from 603/4*/

/** Resets **/
imp_ta.ar = 'b'0;     /* early ta */
stx.ar    = 'b'0;     /* For simulation only */
sty.ar    = 'b'0;     /* For simulation only */
stz.ar    = 'b'0;     /* For simulation only */
ena.ar    = 'b'0;     /* bi-dir buffer and peri */
uds.ar    = 'b'0;     /* uds for peri */
lds.ar    = 'b'0;     /* lds for peri */
as.ar     = 'b'0;     /* as for peri */
rw.ar     = 'b'0;     /* rw for buf and peri */
imp_tea.ar= 'b'0;     /* imp_tea to ppc */

/* Presets */
imp_ta.sp = 'b'0;     /* early ta */
stx.sp    = 'b'0;     /* For simulation only */
sty.sp    = 'b'0;     /* For simulation only */
stz.sp    = 'b'0;     /* For simulation only */
ena.sp    = 'b'0;     /* bi-dir buffer and peri */
uds.sp    = 'b'0;     /* uds for peri */
lds.sp    = 'b'0;     /* lds for peri */
as.sp     = 'b'0;     /* as for peri */
rw.sp     = 'b'0;     /* rw for buf and peri */
imp_tea.sp= 'b'0;     /* imp_tea to ppc */
```

**Listing 2 IMP.H and PPCIMP.C**

```
/* -------------------------------------------------------------------*/
/* IMP (MC68302) definitions - IMP.H                                   */
/* System dependent - For address and databus mappings given in        */
/* applications note                                                   */
/* rev 1.0 jdr/cmd                                                     */
/* --------------------------------------------------------------- */

#define BAR      (0x0F2*4)        /* bar at 0xF2                   */
#define IMP_IPR  (0x814*4)        /* irq pending reg               */
#define IMP_IMR  (0x816*4)        /* irq mask reg                  */
#define ISR      (0x818*4)        /* irq status reg                */
#define PACNT    (0x81e*4)        /* pacnt  equ b+$81e             */
#define PADDR    (0x820*4)
#define PADAT    (0x822*4)
#define PBCNT    (0x824*4)
#define PBDDR    (0x826*4)
#define PBDAT    (0x828*4)
#define ISR      (0x818*4)        /* irq status reg                */
#define WWR      (0x84A*4)        /* watchdog ref reg              */
#define CR302    (0x860*4)        /* cr           equ b+$860       */
#define SCON     (0x882*4)        /* scon         equ b+$882       */
#define SCM      (0x884*4)        /* scm          equ b+$884       */
#define DSR      (0x886*4)        /* dsr          equ b+$886       */
#define SCCE     (0x888*4)        /* scce         equ b+$888       */
#define SCCM     (0x88a*4)        /* sccm         equ b+$88a       */
#define SCCS     (0x88c*4)        /* sccs         equ b+$88c       */
#define SPMODE   (0x8b0*4)        /* spmode       equ b+$8b0       */
#define SIMASK   (0x8b2*4)        /* simask       equ b+$8b2       */
#define SIMODE   (0x8b4*4)        /* simode       equ b+$8b4       */
/* ----------- SCC1 PARAMETER RAM -------------------------------- */
#define RFCR     (0x480*4)        /*                               */
#define TFCR     ((0x480*4)+1)    /*                               */
#define MRBLR    (0x482*4)        /*                               */
/* ------------------ SCC UART Specific PRAM ---------------------- */
#define PAREC    (0x4A2*4)        /* parity error cntr             */
#define FRMEC    (0x4A4*4)        /* framing error cntr            */
#define NOSEC    (0x4A6*4)        /* noise cntr                    */
#define BRKEC    (0x4A8*4)        /* break condn cntr              */
#define MAXID    (0x49c*4)        /* maxid        equ b+$69c       */
#define CHAR1    (0x4b0*4)        /* cntrl charact1                */
#define CHAR8    (0x4BE*4)        /* cntrl character8              */
#define TXBD0    (0x440*4)        /* set up first tx BD            */
#define TXBD02   ((0x440+2)*4)    /*                               */
#define TXBD04   ((0x440+4)*4)    /*                               */
#define TXBD06   ((0x440+6)*4)    /*                               */
#define RXBD00   (0x400*4)        /* rx buff control               */
#define RXBD02   (0x402*4)        /* rx buff length                */
#define RXBD04   (0x404*4)        /* rx buf pointer                */
#define RXBD06   (0x406*4)        /* rx buf pointer                */
#define RXBD10   (0x408*4)        /* rx buff control               */
#define RXBD12   (0x40A*4)        /* rx buff length                */
#define RXBD14   (0x40C*4)        /* rx buf pointer                */
#define RXBD16   (0x40E*4)        /* rx buf pointer                */
/* --------------------------------------------------------------- */
```

```
/********************************************************
Title       :      PPCIMP.C
Description:       C language mc68302 UART demonstration
            :      program. Includes general and sc
            :      specific init + getc and putc
Date        :      06/04/94
Version     :      0.1
Author      :      Colin MacDonald
History     :      Based on programs by John D. Round &
            :      Peter Crooks. Thanks to Gordon Lawton &
            :      Dominic Gallacher for debug assistance
*********************************************************/
#include "impu.h"
```

```
#define UNS8 unsigned char
#define UNS16 unsigned short
#define UNS32 unsigned long
#define NULL 0x00
#define TXB0_OFF 0x0000
#define RXB0_OFF 0x0100
#define RXB1_OFF 0x0102
#define ONECHAR 1
#define MAXRXBSZ 1
#define OUTCHAR '5'
#define ENA_TXRX 0x000C
/* ******************************************************/
/* 302 defines before initialisation of BAR register    */
#define BASE_302_RESET 0x80000000 /* chip select for 302*/
#define BAR_INIT 0x0001             /* +b12  0x4000 ofs */
/* Base addr of all accesses to 302 after BAR init      */
#define BASE_302 (BASE_302_RESET+0x4000)
/*******************************************************/

void  init_impu_gen();
void  init_impu_sc(char);
void  init_duart();
int   putc(char,char);
int   getc(char);
void  put8(UNS32,UNS8);
void  put16(UNS32,UNS16);
UNS16 get16(UNS32);
UNS8  get8(UNS32);

char outchar;
char scn=0;
int  i;
int  c = 'x';

main()
{
  init_impu_gen();
  init_impu_sc(1);
  init_impu_sc(2);
  putc('T',1);
  putc('1',1);
  putc('-',1);
  putc('T',2);
  putc('2',2);
  putc('-',2);
  while (1)
  {
    c=getc(2);
    putc(c,1);
    putc(c,2);
  }
}

void init_impu_gen()
{
  UNS16 bar_reg=0;
  /* Initialise the BAR register bit 12 set to 1        */
  put16((BASE_302_RESET+BAR),BAR_INIT);
  bar_reg = get16(BASE_302_RESET+BAR);
  /* This maps the 302 to offset 0x4000 on off-board    */
  put16(BASE_302+PACNT,0xffff);
  /* PA0=RxD2, PA1=TxD2 PA2-15=PA2-15                   */
  put16(BASE_302+SIMODE,0x000C);
  /* SCC3=MUX, SCC2=UNMUX,  SCC1=UNMUX                  */
}

void init_impu_sc(char scn)
{
  volatile UNS32 internal_ram=0;
  UNS16 t_reg;
  UNS16 rxflg=0x8000;
  UNS32 sroff, spoff;
  sroff = (UNS32) ((scn-1)*4*0x010);
  spoff = (UNS32) ((scn-1)*4*0x100);

  /*----------------------------------------------------*/
  /* GENERAL & UART SPECIFIC PARAMETER RAM INIT         */
  /*----------------------------------------------------*/
  put8(BASE_302+spoff+RFCR,0x00);
  put8(BASE_302+spoff+TFCR,0x00);
  /* used for off-chip bufs simply init to non FFFF val */
  put16(BASE_302+spoff+MRBLR,MAXRXBSZ);  /*rxbuf max sz */
  /* MRBLR defines siz of buf,rbd defines how much used */
  put16(BASE_302+spoff+MAXID,0x0080); /* max idl chrs   */
  /* MAX_IDL #1-65536                                   */
  /* BRKCR - Not written, no STOP TRANSMIT command give */
  put16(BASE_302+spoff+PAREC,0x0000);
  /* Initialise parity error count                      */
  put16(BASE_302+spoff+FRMEC,0x0000);
  /* Initialise FRAMING error count                     */
  put16(BASE_302+spoff+NOSEC,0x0000);
  /* Initialise NOISE error count                       */
  put16(BASE_302+spoff+BRKEC,0x0000);
  /* Initialise BREAK error count                       */
  /* UADDR1/UADDR2 not written - no multidrop mode      */
  put16(BASE_302+spoff+CHAR1,0x8000); /* datasht recmmd */
  put16(BASE_302+spoff+CHAR8,0x0000); /* datasht recmmd */
  /* CHAR2-7 not written - vals for 1/8 disable all     */

  /*----------------------------------------------------*/
  /* SCC INITIALISATION                                 */
  /*----------------------------------------------------*/
  /* NOTE: this divider val is for 302 clck of 16.0MHz! */
  put16(BASE_302+sroff+SCON,0x00CE);
  /* TCS,RCS=BRG,clk div=103 (103*2=0xCE) baud =  9600  */
  /* TCS,RCS=BRG,clk div=51  (51*2=0x66)  baud =  19.2k */
  put16(BASE_302+sroff+SCM,0x01b1);
  /* SCM=UARTM||SCM !parity, normal op, 8-b/c s/w CTS   */
  /* !ENR,!ENT, async UART                              */
  put16(BASE_302+sroff+DSR,0x7000);
  /* norm stop bits                                     */
  put8(BASE_302+sroff+SCCE,0xFF);          /* clear bits */
  put8(BASE_302+sroff+SCCM,0x00);          /*msk all irqs*/

  /* transmit buffer init */
  put16(BASE_302+spoff+TXBD02,0x0);
  put16(BASE_302+spoff+TXBD04,0x0000);     /* this points*/
  put16(BASE_302+spoff+TXBD06,TXB0_OFF);   /*to strng loc*/
  put16(BASE_302+spoff+TXBD0,0x2800);      /* -R,+W,+CR   */

  /* receive buffer init */
  put16(BASE_302+spoff+RXBD04,0x0000);     /* =tx buff + */
  put16(BASE_302+spoff+RXBD06,RXB0_OFF);   /*small offset*/
  put16(BASE_302+spoff+RXBD00,0x2000);     /*-ebit, -wbit*/

  /* check rxflg set before exit */
  while ((rxflg & 0x8000) == 0x8000)
  rxflg = get16(BASE_302+spoff+RXBD00);    /*get rdy stts*/

  /* SCM init */
  t_reg=get16(BASE_302+sroff+SCM);         /* read in reg*/
  t_reg += ENA_TXRX;
  put16(BASE_302+sroff+SCM,t_reg);         /*+ena Rx & Tx*/
}

int putc(char outchar, char scn)
{
  UNS16 txflg=0x0000;
  volatile UNS32 internal_ram=0;
  UNS32 sroff, spoff;
  sroff = (UNS32) ((scn-1)*4*0x010);
  spoff = (UNS32) ((scn-1)*4*0x100);
  /* check buffer is not in use */
  txflg = get16(BASE_302+spoff+TXBD0);
  if ((txflg & 0x8000) == 0x8000)
    return(1);
  else
  {
    txflg=0x8000;
    /* Put a char into internal RAM */
    put8(BASE_302+(internal_ram*4),outchar);
    put16(BASE_302+spoff+TXBD02,ONECHAR);
    put16(BASE_302+spoff+TXBD04,0x0000);  /*this points */
    put16(BASE_302+spoff+TXBD06,0x0000);  /*to strng loc*/
    put16(BASE_302+spoff+TXBD0,0xA800);   /*int tx !CTS */
    while ((txflg & 0x8000) == 0x8000)
    txflg = get16(BASE_302+spoff+TXBD0);  /*get rdy stts*/
    return(0);
  }
}

int getc(char scn)
{
  UNS8  rxdta;
  UNS16 rxflg=0x8000;
  UNS16 t1;
  volatile UNS32 internal_ram=0;
  UNS32 sroff, spoff;
  sroff = (UNS32) ((scn-1)*4*0x010);
  spoff = (UNS32) ((scn-1)*4*0x100);

  rxflg = get16(BASE_302+spoff+RXBD00);   /*get rdy stts*/
  if ((rxflg & 0x8000) == 0x8000)
  return (1);
  else
  {
    /* receive buffer init */
    put16(BASE_302+spoff+RXBD04,0x0000);  /* =tx buff + */
    put16(BASE_302+spoff+RXBD06,RXB0_OFF);/*small offset*/
    put16(BASE_302+spoff+RXBD00,0xA000);  /*+ebit, -wbit*/
    rxflg = 0x8000;
    while ((rxflg & 0x8000) == 0x8000)
    rxflg = get16(BASE_302+spoff+RXBD00); /*get rdy stts*/
    rxdta=get8(BASE_302+((RXB0_OFF+internal_ram)*4));
    return(rxdta);
  }
}

void put8(UNS32 address, UNS8 output_data)
{
  volatile UNS8 *memory;
  memory=(UNS8 *)address;
  *memory=output_data;
}

UNS8 get8(UNS32 address)
{
  UNS8 *memory,output_data;
  memory=(UNS8 *)address;
  output_data=*memory;
  return output_data;
}

void put16(UNS32 address, UNS16 output_data)
{
  volatile UNS16 *memory;
  memory=(UNS16 *) address;
  *memory=output_data;
}

UNS16 get16(UNS32 address)
{
  volatile UNS16 read_data;
  read_data = *(UNS16 *) address;
  return read_data;
}
```

## *How to Reach Us:*

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only:***
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com