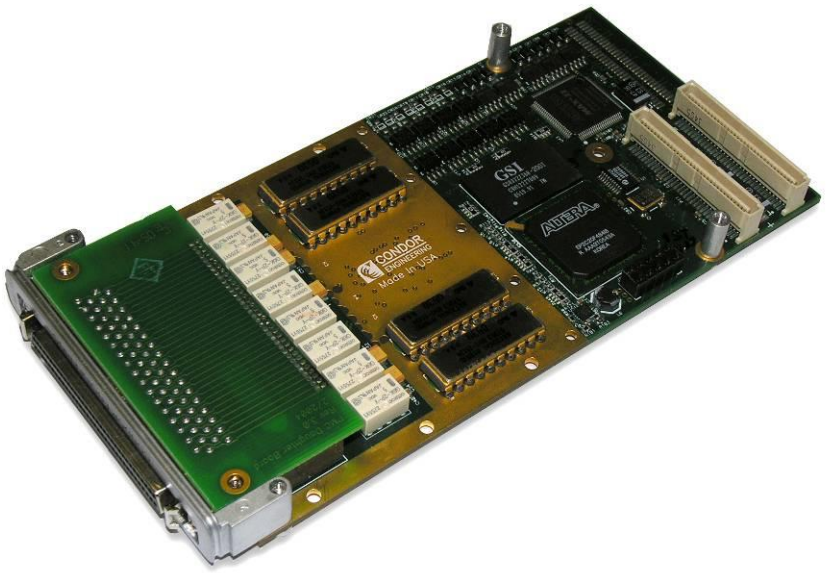
Application Note AN-016

Setting a Bus Controller Watchdog Timer using Conditional Branching



abaco
SYSTEMS

Copyrights

Abaco Systems, Inc.
26 Castilian Drive, Suite B
Goleta, CA 93117
Main +1 805-965-8000 or +1 805- 883-6101
Support +1 805-965-8000 or +1 805- 883-6097

support@abaco.com (email)
https://www.abaco.com/products/avionics

Additional Resources

For more information, please visit the Abaco Systems website at:

www.abaco.com

Setting up a Bus Controller watchdog

The Bus Controller normally runs in continuous mode, cycling through the messages in minor-frames. Once this continuous mode starts, the host processor must stop the Bus Controller. If the host controller crashes, the Bus Controller continues to transmit until the user powers off to the 1553 board. To prevent this condition the user can create a watchdog that will stop the Bus Controller if the host processor goes down.

The Bus Controller sets up messages to run continuously by creating minor-frames. These frames are made up of messages the BC will transmit. Each message points to the next message to execute. The last message in a frame, normally points back the first message. This creates the continual loop that runs until stopped by the host. In addition to BC messages, the frames can also contain conditional branch commands. These commands branch the execution of the frame messages depending on pre-specified conditions. Conditions can either be a value in board memory or a count.

Branching on count allows the user's application to set up a branch to execute on a particular count value. The branch-counter decrements the count value until zero and then branches to the specified command. For example, a count value of two causes the execution to branch on every third time the branch executes.

To create a watchdog the user must set up a branch on count that branches to a BC HALT command. During execution, the host must periodically reset the count to prevent it from reaching zero. If the host crashes during execution, the count is no longer reset. When the count reaches zero the BC controller will execute and BC HALT command stopping the Bus Controller.

The following coding example shows how to create a watch dog timer. There is a branch on count at the end of the frame followed by a BC_CONTROL_HALT and a BC_CONTROL_NOP. This conditional branch branches every third frame. The BC_CONTROL_HALT stops the Bus Controller and disables any BC triggering. If the branch count value reaches zero (0), the BC_CONTORL_HALT command is executed and the BC will stop. If the user application resets the count prior reaching zero, the frame continues to run.

```
int BusController_Init(int cardnum,int num_buffers)
{
   int status, messno,i;
   API_BC_MBUF bcmessage;
   memset(retry,0,sizeof(BT_U16BIT)*9);

   status=BusTools_BC_Init(cardnum,0,BT1553_INT_END_OF_MESS,0,16,14, 10000, 1);
   printf("BusTools_BC_Init status = %d\n",status);

   status = BusTools_BC_MessageAlloc(cardnum,50);//num_buffers);
   printf("BusTools_BC_MessageAlloc status = %d\n",status);

   messno = 0;
   memset((char*)&bcmessage,0,sizeof(bcmessage));
   bcmessage.messno = messno;
   bcmessage.messno_next = (BT_U16BIT)(messno + 1);
   bcmessage.control = BC_CONTROL_MESSAGE;        // show as a message
   bcmessage.control |= BC_CONTROL_BUFFERA;       // use buffer A
   bcmessage.control |= BC_CONTROL_CHANNELA;      // Transmit on Channel A
   bcmessage.control |= BC_CONTROL_INTERRUPT;
```

```
bcmessage.control |= BC_CONTROL_MFRAME_BEG;    //Start of a Minor Frame
bcmessage.mess_command1.rtaddr   = 1;
bcmessage.mess_command1.subaddr  = 2;
bcmessage.mess_command1.wcount   = 2;
bcmessage.mess_command1.tran_rec = 0;
bcmessage.errorid = 0;       // Default error injection buffer (no errors)
bcmessage.gap_time = 10;     // 10 microsecond inter-message gap.

for(i=0;i<32;i++)
    bcmessage.data[0][i] = 0xA000+i;

status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
printf("BusTools_BC_MessageWrite status = %d\n",status);

messno++;
memset((char*)&bcmessage,0,sizeof(bcmessage));
bcmessage.messno = messno;
bcmessage.messno_next = (BT_U16BIT)(messno + 1);
bcmessage.control = BC_CONTROL_MESSAGE;        // show as a message
bcmessage.control |= BC_CONTROL_BUFFERA;       // use buffer A
bcmessage.control |= BC_CONTROL_CHANNELA;      // Transmit on Channel A
bcmessage.control |= BC_CONTROL_INTERRUPT;
bcmessage.mess_command1.rtaddr   = 3;
bcmessage.mess_command1.subaddr  = 4;
bcmessage.mess_command1.wcount   = 4;
bcmessage.mess_command1.tran_rec = 1;

bcmessage.errorid = 0;     // Default error injection buffer (no errors)
bcmessage.gap_time = 60;   // 60 microsecond inter-message gap.

status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
printf("BusTools_BC_MessageWrite status = %d\n",status);

messno++;
memset((char*)&bcmessage,0,sizeof(bcmessage));
bcmessage.messno = messno;
bcmessage.messno_next = (BT_U16BIT)(messno + 1);
bcmessage.control = BC_CONTROL_MESSAGE;        // show as a message
bcmessage.control |= BC_CONTROL_BUFFERA;       // use buffer A
bcmessage.control |= BC_CONTROL_CHANNELA;      // Transmit on Channel A
bcmessage.control |= BC_CONTROL_INTERRUPT;
bcmessage.mess_command1.rtaddr   = 1;
bcmessage.mess_command1.subaddr  = 8;
bcmessage.mess_command1.wcount   = 8;
bcmessage.mess_command1.tran_rec = 1;
bcmessage.errorid  = 0;     // Default error injection buffer (no errors)
bcmessage.gap_time = 100;   // 100 microsecond inter-message gap.

status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
printf("BusTools_BC_MessageWrite status = %d\n",status);

messno++;
memset((char*)&bcmessage,0,sizeof(bcmessage));
bcmessage.messno = messno;
bcmessage.messno_next = (BT_U16BIT)(messno+2);    // Next to a noop
bcmessage.messno_branch = (BT_U16BIT)(messno+1);  // Branch to a stop
bcmessage.control = BC_CONTROL_CONDITION2;        // Branch command
bcmessage. cond_counter    = 2;                   // Initial count
bcmessage. .cond_count_val = 2;                   // Go every third frame.

status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
printf("BusTools_BC_MessageWrite status = %d\n",status);
```

```
   messno++;
   memset((char*)&bcmessage,0,sizeof(bcmessage));
   bcmessage.messno = messno;
   bcmessage.messno_next = (BT_U16BIT)(0);     //
   bcmessage.control = BC_CONTROL_HALT;        // Stop the BC

   status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
   printf("BusTools_BC_MessageWrite status = %d\n",status);

   messno++;
   memset((char*)&bcmessage,0,sizeof(bcmessage));
   bcmessage.messno = messno;
   bcmessage.messno_next = (BT_U16BIT)(0);         //
   bcmessage.control = BC_CONTROL_NOP;
   bcmessage.control |= BC_CONTROL_INTERRUPT;      // Setup interrupt
   bcmessage.control |= BC_CONTROL_MFRAME_END;     //End of a Minor Frame

   status = BusTools_BC_MessageWrite(cardnum,messno,&bcmessage);
   printf("BusTools_BC_MessageWrite status = %d\n",status);

   return status;
}
```

Once the conditional branch on count is in place, the host must periodically reset the count value to prevent the BC HALT command from executing. The following code shows how the host resets the count value.

First you must get the address of the BC Conditional branch message buffer. This is done once during initialization.

```
status = BusTools_BC_MessageGetAddr(cardnum,messno,&mess_addr);
mess_addr = mess_addr+12 //byte offset to the counter value
```

Then periodically reset the counter back to the original value.

```
BT_U16BIT reset_value = 3;
Status = BusTools_MemoryWrite2(cardnum,RAM,mess_addr,2,&reset_value);
```

The application must execute the above code a rate sufficient to keep the counter from reaching 0. The application can have this code run off a timer or as part of a user-callback function using BusTools_RegisterFunction.

The following coding example shows how to use BusTools_RegisterFunction to reset the watchdog timer count. The function setupThe_BC_CTL_IntFIFO creates a callback to the bc_intCtlFunction when ever a BC Control event occurs. In the code above, interrupts are enabled on the BC_CONTROL_NOP at the end of the minor-frame. That will reset the counter at the end of each frame.

```
void setupThe_BC_CTL_IntFIFO(int cardnum)
{
   int status;
   // Setup the FIFO structure for this board.
   memset(&sIntFIFO1, 0, sizeof(sIntFIFO1));
   sIntFIFO1.iNotification  =  0;
   sIntFIFO1.function       =  bc_intCtlFunction;
   sIntFIFO1.iPriority      =  THREAD_PRIORITY_ABOVE_NORMAL;
   sIntFIFO1.dwMilliseconds =  INFINITE;
   sIntFIFO1.bForceStartup  =  0;
```

```
   sIntFIFO1.nUser[0]       =  mess_addr;
   sIntFIFO1.FilterType     =  EVENT_BC_CONTROL;

   // Call the register function to register and start the thread.
   status = BusTools_RegisterFunction(cardnum, &sIntFIFO1, 1);
   printf("BusTools_RegisterFunction status = %d\n",status);
}
```

When the Noop command is executed the bc_intCtlFuntion executes and resets the branch-count. Make sure mess_addr, calculated during initialization, is global or passed in the interrupt FIFO as user data.

```
BT_INT  _stdcall bc_intCtlFunction(BT_UINT cardnum, struct api_int_fifo
*sIntFIFO)
{
   BT_INT tail;
   BT_U16BIT reset_value = 3;

   tail = sIntFIFO->tail_index;
   mess_addr = sIntFIFO->nUser[0];
   while(tail != sIntFIFO->head_index)
   {
      if(sIntFIFO->fifo[tail].event_type & EVENT_BC_CONTROL)
      {
         Status = BusTools_MemoryWrite(cardnum,mess_addr,2,&reset_value)
      }
      tail++;
      tail &= sIntFIFO->mask_index;
      sIntFIFO->tail_index = tail;
   }
   return 0;
}
```