Appendix A. Code Examples

Bounded-Queue Example Code

Variables and Constants

```
// These just make life easier
typedef unsigned char    u8;
typedef unsigned short   u16;
typedef unsigned int     u32;
typedef signed char      s8;
typedef short            s16;
typedef int              s32;

#define C_Q_MAX         ( 32 )
#define C_Q_MAX_ID      ( C_Q_MAX * 2 )
#define C_Q_DATASIZE    ( 31 )
#define C_Q_SIZE        ( 64 )
#define C_Q_FULL        ( -1 )
#define C_Q_XSUM_ERROR  ( -2 )
#define C_FLASH_EMPTY   ( 0xFF )

typedef struct {
    u16 iData[C_Q_DATASIZE];
    u8 iID;
    u8 iChecksum;
} QENTRY;

extern bool ChecksumValid(QENTRY *pEntry);
extern u16 flashEraseSector(void *pAddress);
extern u16 flashWrite(void *pAddress, u16 iData);

extern QENTRY FlashQueue[C_Q_MAX]; // Mapped into data flash
extern u8 iQID;
extern u8 iQIndex;
```

Initialization Code

```
/*
// queueInitialize()
//    This routine returns the current valid entry in the queue and
//    sets the global variable iQID to the ID of the current valid entry.
*/
short queueInitialize(void)
{
s16 iIndex;
    iQID = 0;
    // Only the current sector will have the first entry used
    // and the last entry empty.
    if (FlashQueue[0].iID != C_FLASH_EMPTY &&
        FlashQueue[C_Q_MAX-1].iID == C_FLASH_EMPTY)
    {
        // Find the last valid entry
        for (iIndex=1;iIndex < C_Q_MAX-1; ++iIndex)
        {
            if (FlashQueue[iIndex].iID == C_FLASH_EMPTY)
```

```
            {
                iQID = FlashQueue[iIndex-1].iID;
                return iIndex - 1;
            }
        }
    }
     // Should never get here.  The queue is full, return error
    return C_Q_FULL;
}
```

Write-Queue Entry Code

```
/*
// queueWriteEntry()
//    Writes new entry into the flash queue.
*/
short queueWriteEntry(QENTRY *pEntry)
{
void *pDest;
u8 i;
    if (pEntry != NULL)
    {
        // Compute the new packet ID
        iQID = (iQID + 1) % C_Q_MAX_ID;

        // Compute the new Index
        iQIndex = (iQIndex + 1) % C_Q_MAX;

        // Set the packet ID
        pEntry->iID = iQID;

        // Calculate the checksum
        pEntry->iChecksum = iQID;
        for (i = 0; i < C_Q_DATASIZE; ++i)
            pEntry->iChecksum += pEntry->iData[i];

        // If next entry is already full erase entire sector
        if (FlashQueue[iQIndex].iID != C_FLASH_EMPTY)
            flashEraseSector(FlashQueue);

        // Write packet to flash
        pDest = &FlashQueue[iQIndex];
        for (i = 0; i < sizeof(QENTRY)/2; ++i)
            flashWrite(pDest, ((u16 *)pEntry)[i]);
        return true;
    }
    return false;
}
```

Bank-Switching Example Code

Variables and Constants

```
#define C_B_BANKS       ( 2 )
#define C_B_SIZE        ( 256 )
#define C_B_DATASIZE    ( 255 )
#define C_B_FULL        ( -1 )
#define C_B_MAX_ID      ( C_B_BANKS * 2 )
#define C_B_XSUM_ERROR  ( -2 )
#define C_FLASH_EMPTY   ( 0xFF )

typedef struct {
    u16 iData[C_B_DATASIZE];
    u8  iID;
    u8  iChecksum;
} FLASHBANK;

extern bool ChecksumValid(FLASHBANK *pBank);
extern u16 flashEraseSector(void *pAddress);
extern u16 flashWrite(void *pAddress, u16 iData);

extern FLASHBANK FlashBank0; // This needs to be mapped to 0xF000-0xF7FF
extern FLASHBANK FlashBank1; // This needs to be mapped to 0xE000-0xE7FF
extern u8 iBank;
extern u8 iBID;

FLASHBANK *FlashBanks[C_B_BANKS] = {&FlashBank0, &FlashBank1};
```

Initialization Code

```
/*
// bankInitialize()
//    This routine sets iBank to the current valid sector.
*/
u8 bankInitialize()
{
u16 i;
    iBank = 0;
    for (i = 0; i < C_B_BANKS; ++i)
    {
        if (((FlashBanks[i]->iID + 1)%128) != FlashBanks[(i+1)%C_B_BANKS]-
>iID)
        {
            iBank = i;
            break;
        }
    }
    return iBank;
}
```

Write Bank Code

```
/*
// bankWrite()
//    Writes new data bank into the next available sector.
*/
bool bankWrite(FLASHBANK *pBank)
{
u16 i;
    if (pBank != NULL)
    {
        // Compute the new packet ID
        iBID = (iBID + 1) % C_B_MAX_ID;

        // Compute the new bank number
        iBank = (iBank + 1) % C_B_BANKS;

        // Set the bank ID
        pBank->iID = iBID;

        // Calculate the checksum
        pBank->iChecksum = iBID;
        for (i = 0; i < C_B_DATASIZE; ++i)
            pBank->iChecksum += pBank->iData[i];

        // If bank not empty erase it
        if (FlashBanks[iBank]->iID != C_FLASH_EMPTY)
            flashEraseSector(&FlashBanks[iBank]);
        // Write bank to flash sector
        for (i = 0; i < sizeof(FLASHBANK)/2; ++i)
            flashWrite((&(FlashBanks[iBank]))[i], ((u16 *)pBank)[i]);
        return true;
    }
    return false;
}
```

Bounded-Queue and Bank-Switching Example Code

Variables and Constants

```
#define C_Q_BANKSIZE      ( 32 )
#define C_Q_BANKS         (  2 )
#define C_Q_ERASE_THRESH  ( C_Q_BANKSIZE - 4 )
#define C_Q_MAX           ( C_Q_BANKSIZE * C_Q_BANKS )
#define C_Q_MAX_ID        ( C_Q_MAX * 2 )
#define C_Q_DATASIZE      ( 31 )
#define C_Q_ENTRY_SIZE    ( 64 )
#define C_Q_FULL          ( -1 )
#define C_Q_XSUM_ERROR    ( -2 )
#define C_FLASH_EMPTY     ( 0xFF )

typedef struct {
    u16 iData[C_Q_DATASIZE];
    u8 iID;
    u8 iChecksum;
} QENTRY;

extern bool ChecksumValid(QENTRY *pEntry);
extern u16 flashEraseSector(void *pAddress);
extern u16 flashWrite(void *pAddress, u16 iData);

extern QENTRY FQueueBank0[C_Q_MAX]; // Mapped to data flash sector 0
extern QENTRY FQueueBank1[C_Q_MAX]; // Mapped to data flash sector 1
extern u8 iQID;                     // ID number of the last entry
extern u8 iQIndex;                  // Index of current entry
extern u8 iBank;                    // Index of current flash bank

QENTRY *FQueue[C_Q_BANKS] = {FQueueBank0, FQueueBank1};
```

Initialization Code

```
/*
// queueInitialize()
//    This routine returns the current valid entry in the queue and
//    sets the global variable iQID to the ID of the current valid entry.
*/
short queueInitialize(void)
{
s16 iIndex;
    iQID = 0;
    // Find the active sector
    for (iBank=0;iBank < C_Q_BANKS; ++iBank)
    {
        // Only the current sectore will have the first entry used
        // and the last entry empty.
        if (FQueue[iBank][0].iID != C_FLASH_EMPTY &&
            FQueue[iBank][C_Q_BANKSIZE-1].iID == C_FLASH_EMPTY) {

            // Find the last valid entry
            for (iIndex=1;iIndex < C_Q_BANKSIZE-1; ++iIndex)
            {
                if (FQueue[iBank][iIndex].iID == C_FLASH_EMPTY)
                {
                    iQID = FQueue[iBank][iIndex-1].iID;
                    return iIndex - 1 + C_Q_BANKSIZE*iBank;
                }
```

```
            }
        }
    }

    // Should never get here.  The queue is full, return error
    return C_Q_FULL;
}
```

Write Data Code

```
/*
// queueWriteEntry()
//    Writes new entry into the flash queue.
*/
short queueWriteEntry(QENTRY *pEntry)
{
void *pDest;
u16 i;
    if (pEntry != NULL)
    {
        // Compute the new packet ID
        iQID = (iQID + 1) % C_Q_MAX_ID;

        // Compute the new Index
        iQIndex = (iQIndex + 1) % C_Q_MAX;

        // Set the packet ID
        pEntry->iID = iQID;

        // Calculate the checksum
        pEntry->iChecksum = iQID;
        for (i = 0; i < C_Q_DATASIZE; ++i)
            pEntry->iChecksum += pEntry->iData[i];

        // Write packet to flash
        pDest = &FQueue[iQIndex / C_Q_BANKSIZE][iQIndex % C_Q_BANKSIZE];
        for (i = 0; i < sizeof(QENTRY)/2; ++i)
            flashWrite(pDest, ((u16 *)pEntry)[i]);
        return true;
    }
    return false;
}
```

Erase Sector Maintenance Code

```
/*
// queueEraseSectorMaintenance()
//    This routine monitors the flash sectors and when the current sector is
//    almost full it erases the next sector to free up more room.
//    Returns true if a sector erase occurred during the call.
//
//    This routine must be called at least once per queueWriteEntry() call.
*/
bool queueEraseSectorMaintenance(void)
{
    // if the sector is almost full then we want to try and erase the next
sector
    if ((iQIndex % C_Q_BANKSIZE) > C_Q_ERASE_THRESH)
```

```
    {
        short iBank = (iQIndex / C_Q_BANKSIZE + 1) % C_Q_BANKS;

        // If there are restrictions on when flash can be erased
        // then additional conditional can be added here.
        // Just make sure that the next sector is erased before the current
        // sector is full.
        if (FQueue[iBank][0].iID != C_FLASH_EMPTY)
        {
            flashEraseSector(FQueue[iBank]);
            return true;
        }
    }
    return false;
}
```