SilentSystem

ＯＳ－１ユーザーズマニュアル

Ver1.01　2007/8/10

はじめに

この度はサイレントシステムのワンチップサーバーＯＳ－１をご購入頂きありがとうございました。ＯＳ－１はユーザーがそれぞれの用途に応じたプログラムを書き込んで動作させる組み込みシステム向けの超小型ネットワークサーバーです。使用に当ってはまず本マニュアルをよくお読みになり十分に理解した上でご利用ください。

免責事項

ＯＳ－１は一般電子機器用の半導体部品を使用しておりますので、生命に関わる用途や身体に害を及ぼす恐れのある用途には使用出来ません。またＯＳ－１は基本的にお客様が使用目的に適合したソフトウェアを組み込んで使用する機器ですので、使用の前に十分なテストを行い正しく動作を確認してから使用を開始して下さい。またＯＳ－１の運用の結果については有限会社サイレントシステムはいかなる責任も負えません。

ＯＳー１に内蔵しているソフトウェア及びマニュアルには欠陥が含まれている可能性がありますので、その信頼性や正確性を保証する事は出来ません。またその欠陥を修正する事を保証する事もできません。

取扱上の注意

電源が入った状態でいかなるコネクタにも着脱をしないで下さい。最悪の場合ラッチアップを起こして半導体を破損する可能性があります。

ＯＳ－１は静電気により内部の半導体が破損する可能性があります。くれぐれも静電気対策に配慮してください。

外部から大きなノイズやサージをＯＳ－１に与えると内部の半導体がラッチアップを起こして半導体を破損する可能性があります。入出力ポートや電源にノイズやサージが混入したり電源の電圧が急激に変動しないように使用してください。

IOポートやEthernetへは規定された信号レベルを接続して下さい。また信号の極性を間違えると半導体を破損する可能性があります。

衝撃や振動や衝突などの強い衝撃を与えないでください。ショックでＯＳ－１の内部を破損する可能性があります。

動作環境は極端な高低温や多湿を避けて規定された環境でご利用ください。また塵埃の多い環境で使用すると電流がリークして半導体を破損する可能性があります。

その他社会的通念上一般的な電子機器の動作にとって支障のある環境での利用は避けて下さい。

ＯＳ－１の概要

ＯＳ－１は米国FreeScale社の半導体MC9S12NE64を活用したワンチップサーバーです。簡単にユーザープログラムを開発できますのでネットワークを利用した組み込み機器向けに最適なモジュールです。ＯＳ－１には以下の特徴があります。

・自社製のオリジナルＯＳ「SilentMoon」を搭載しています

SilentMoonはユーザーがＯＳ－１を利用してユーザープログラムを作成する際のベースとなるＯＳです。マルチスレッド、ＴＣＰ／IPスタック、メモリー管理、ファイルシステム、レジストリなどの組み込み用途に便利な機能を満載しています。

・Ｃ言語インタープリター「SilentC」を搭載しています

ユーザープログラムの開発を容易にするために工場出荷状態でＣ言語インタープリターを内蔵しています。ＯＳ－１単体でユーザープログラムを開発できます。シリアルポートあるいはTelnetを利用して会話型に動作テストをしながら開発とデバッグが進めらます。またＯＳ－１を機器を組み込んだ後でも内部のプログラムを現場で容易に変更可能です。

・本体のみでフラッシュメモリーのプログラムが可能です

弊社のサイトからサンプルアプリケーションをダウンロードしてそのファイルをＯＳ－１に転送するだけで起動時に自動的にフラッシュメモリーを消去して指定のファイルをプログラムします。この機能を利用して最新のファームウェアや他のユーザーが開発したプログラムを利用できます。

・工場出荷状態へのリカバリ機能を搭載しています

フラッシュやファイルシステムを誤って消去してしまったり誤ったファイルを書き込んでしまった場合にもユーザーがアクセスできないリカバリ領域から工場出荷状態のデータを読み出して初期化する機能を搭載していますので安心してユーザープログラムを書き込めます。

まず動作させてみる

まずＯＳ－１を動作させてみましょう。以下のものを準備して下さい。

・ＯＳＩＯ－１

ＯＳ－１は電源を供給すれば単体でも動作するように設計されていますが、ユーザープログラムの開発や書き込み、最新ファームウェアへのアップデートなどにはＯＳＩＯ－１を利用することを想定しています。こうした事情から本書ではＯＳＩＯ－１との組み合わせを前提とした解説を記しております。ＯＳ－１はＯＳＩＯ－１に正しく取り付けて下さい。ＯＳ－１の取り付けが正常でないと通電した際にＯＳ－１を破損する可能性があります。

・専用ＡＣアダプタ

ＯＳ－１は３．３Ｖの電源電圧で動作します。安定した電源を供給するために必ず専用ＡＣアダプターをご利用下さい。その他のＡＣアダプタを利用した場合にはＯＳＩＯ－１及びＯＳ－１本体を破損する可能性があります。

・パソコン

ＯＳ－１の初期設定をしたりユーザープログラムを作成するためにはパソコンが必要です。パソコンの機能としてはシリアルポートが利用可能であること、またイーサネットが利用可能である事が必須です。本書ではWindowsXPがインストールされているパソコンを前提に解説していますがMACやLinuxなどのOSでも構いません。またターミナルソフトも必要になります。

・シリアルケーブル

パソコン本体からD-SUB9ピンのストレートシリアルケーブルでＯＳＩＯ－１に接続します。ハードフローに対応していますのでフル結線のケーブルをお使い下さい。ＯＳＩＯ－１はモデムと同じ極性でパソコンに接続されます。

・イーサネットケーブル

パソコン本体から直接ＯＳ－１にイーサネットを接続する際にはクロスケーブルが必要です。またスイッチングハブを利用する場合にはこれとは逆にストレートケーブルでＯＳ－１とハブを接続します。ＯＳ－１のイーサネット出力はパソコンと同じ極性です。接続した際にＯＳ－１のイーサネットコネクタのオレンジ色のＬＥＤが点灯すれば接続の極性は正常です。100Mbpsがデフォルトのスピードです。

シリアルポートで操作する

パソコンとＯSIＯ－1をストレートシリアルケーブルで接続します。もしシリアルターミナルが用意できない場合には第二段階のネットワーク接続に進んでも結構です。

・シリアルポートの設定をする

WindowsXPに付属しているハイパーターミナルやTeraTermなどのターミナルソフトを起動して通信パラメータを57600bps、データ8ビット、ストップ1ビット、パリティーなし、ハードフローに設定します。この設定はＯＳ－1のデフォルト設定ですのでターミナルソフト側で設定を保存しておくと良いでしょう。

・電源を入れる

ＯSIＯ－1に専用ＡＣアダプタのジャックを挿し込みます。ＯＳ－1とＯSIＯ－1に取り付けられている電源ＬＥＤが両方とも点灯するはずです。もし点灯しない場合にはハード的になんらかの異常の可能性があります。

ＯＳ－1が正しく取り付けられているか確認してください。またＯSIＯ－1には1Ａのヒューズが取り付けられています。過電流が一度でも流れるとヒューズが切れますのでご注意下さい。正常に起動するとターミナルに起動メッセージとOKが表示されます。

・テストプログラムの実行

工場出荷状態のＯＳ－1には簡単なサンプルプログラムが書き込まれています。ターミナルからrunと打ち込みEnterキーを押すとＯSIＯ－1の4つのＬＥＤがフラッシュしながら2つのリレーが動作します。type Mainと打ち込みEnterキーを押すとＣ言語で書かれたサンプルプログラムのソースリストを表示します。ここまで動作すればＯＳ－1及びＯSIＯ－1のセットアップの第一段階は終了です。

LANで操作する

・ネットワークの設定をする

ＯＳ－１は出荷時には192.168.1.10というIPアドレスにセットされています。このIPアドレスに正常にアクセスできるようにするためにパソコンのローカルエリア接続の設定を行わなくてはなりません。ＯＳ－１とネットワーク接続できない原因はこの設定に失敗している事が殆どですので確実に設定してください。

まずはマニュアルでIPアドレスを設定します。「コントロールパネル」の「ネットワーク接続」を開いて「ローカルエリア接続」を右クリックして「プロパティ」を選びます。「インターネット接続(TCP/IP)」を選びプロパティーボタンをクリックします。「次のIPアドレスを使う」をクリックしてマニュアルでIPアドレスを以下のように設定してOKボタンをクリックします。

IPアドレス=192.168.1.2、サブネットマスク=255.255.255.0、その他は空欄でＯＫ

この設定をする事でLANを通じてを192.168.1.1から192.168.1.254までの範囲にアクセスできるようになります。ＯＳ－１の工場出荷状態のＩＰアドレスは192.168.1.10ですのでこのＩＰアドレスにアクセスできるようにするのが設定の第一段階です。

すでにパソコンが利用しているIPアドレス空間が192.168.1.XXXであれば192.168.1.10というIPアドレスが空いているかどうか確認してください。コマンドプロンプトからPING 192.168.1.10というコマンドを打ち込む事で確認できます。もしすでに192.168.1.10が存在していた場合には予めすでにある192.168.1.10というIPアドレスを持つ機器のIPアドレスを他のアドレスに変更して下さい。

・ＯＳ－１と接続します

ＯＳ－１とパソコンをクロスLANケーブルで直接接続します。もしクロスLANケーブルが無ければハブを利用して接続する事になります。ＯＳ－１とパソコンそれぞれをストレートLANケーブルでハブに接続します。その他の機器は同じハブに接続しないで下さい。ＯＳ－１のネットワーク設定が終了するまではパソコンと1対1で接続します。

うまく接続できるとWindowsXPは画面の右下にあるアイコンでローカルエリア接続の準備が出来た旨を表示します。

・ブラウザでアクセスします

InternetExplorerなどのブラウザを起動します。LANにはＯＳ－１しか接続されていませんのでブラウザの起動画面は「ページが表示できません」などのエラーが表示されますが正常な動作です。改めてアドレス欄にhttp://192.168.1.10/と打ち込みEnterを押します。設定画面が表示されればネットワーク接続は成功です。

・telnetで接続します

ＯＳ－１は出荷時にはtelnetサーバーが有効になっていますのでシリアルターミナルが無くても同様の操作が出来ます。ハイパーターミナルやTeraTermなどのターミナルソフトを利用して192.168.1.10に対してtelnet接続を試みて下さい。telnet接続に成功すると起動メッセージとOKが表示されます。runとコマンドを打ち込んでEnterキーを押すとＯＳIＯ－１の４つのＬＥＤがフラッシュしながら２つのリレーが動作します。type Mainと打ち込みEnterキーを押すとＣ言語で書かれたサンプルプログラムのソースリストを表示します。

・最終設定

必要に応じて既存のLANアドレス空間にＯＳ－１が共存できるようにブラウザでIPアドレス設定を行って下さい。設定はリセット後に有効になります。ＯＳ－１のIPアドレスをDHCPを用いてルーターから取得する場合にはルーターを正しく設定して毎回同じIPアドレスを取得するようにして下さい。またＯＳ－１がDHCPサーバーとしてIPアドレスを提供する事も可能ですがこの機能はＯＳ－１とパソコンをクロスLANケーブルで直接接続する場合にのみ有効にしてください。OS-1をLAN内に接続している場合はDHCPサーバー機能を有効にしないで下さい。

変更したIPアドレスを忘れてしまったり設定に失敗してアクセスできなくなってしまったら、SW2を押しながらリセットするとIPアドレスを強制的に192.168.1.10に設定して起動できます。あとはWebページからIPアドレス設定ページにアクセスして正しい値を設定してください。

・出荷状態に戻すには

いろいろ設定を変更したりフラッシュのユーザーエリアにプログラムを書き込んだりファームウェアのアップデートに失敗したりした場合に工場出荷状態に戻すことが可能です。方法はＯSIＯ－１のＳＷ１を押しながらリセットします。そのままの状態で5秒間ＳＷ１を押したまま保持するとリストア処理が開始されます。ＯＳ－１上のＬＥＤが細かく点滅した後に連続点灯に変わり最終的には消灯します。この処理には約１７秒かかります。

ＯＳ－１の構造(ハード)

ＯＳ－１はユーザーが自由にプログラムを書き込んでネットワークに関する様々な用途に対応できるワンチップサーバーです。この機能をフルに活用するためにはある程度ＯＳ－１の内部の構造を理解しておく必要があります。この章ではＯＳ－１の内部の構造に関して解説します。

・ハード構成

ＯＳ－１は米国FreeScale社の半導体MC9S12NE64を利用したワンチップ構成のサーバーです。MC9S12NE64は25MHzのクロックで動作する16ビットのMPUコアと64Kバイトのフラッシュメモリーと8キロバイトのRAMを内蔵しています。

またイーサネットのPHYとMAC部分も同一チップに内蔵していますのでワンチップでイーサネットに直接接続できるマイクロコントローラーです。また各種設定を記憶したりユーザーのデータを格納するためのファイルシステム用に1メガバイトのシリアルフラッシュメモリーも装備しています。

MC9S12NE64の殆どすべてのポートはＯＳ－１の外部に取り出せるように設計してあります。

ＯＳ－１の構造(ソフト)

ＯＳ－１の内部は以下のブロックによって構成されています。

・Bootモニター

Bootモニターはリセット後に最初に動作するプログラムです。シリアル接続を利用してメモリーをダンプしたりフラッシュメモリーを消去･プログラムする機能を提供しています。また工場出荷状態に復帰するリストア機能も提供しています。Bootモニター部分は書き込みプロテクションにより書き換えが出来ませんがその他のメモリーはすべてBootモニターで消去･プログラム可能です。この機能を利用してファームウェアーのアップデートを行います。

・SilentMoon

SilentMoonはサイレント社製のオペレーティングシステム(OS)でユーザーに各種サービスを提供します。システムコールを呼び出すことでマルチスレッド機能、TCP/IP、UDPなどのネットワークスタック、ファイルシステム、メモリ管理などのサービスを提供します。組み込み機器に求められるサービスの殆どを実装していますのでユーザーが容易に専用プログラムを作成することが出来ます。

・ユーザープログラムエリア

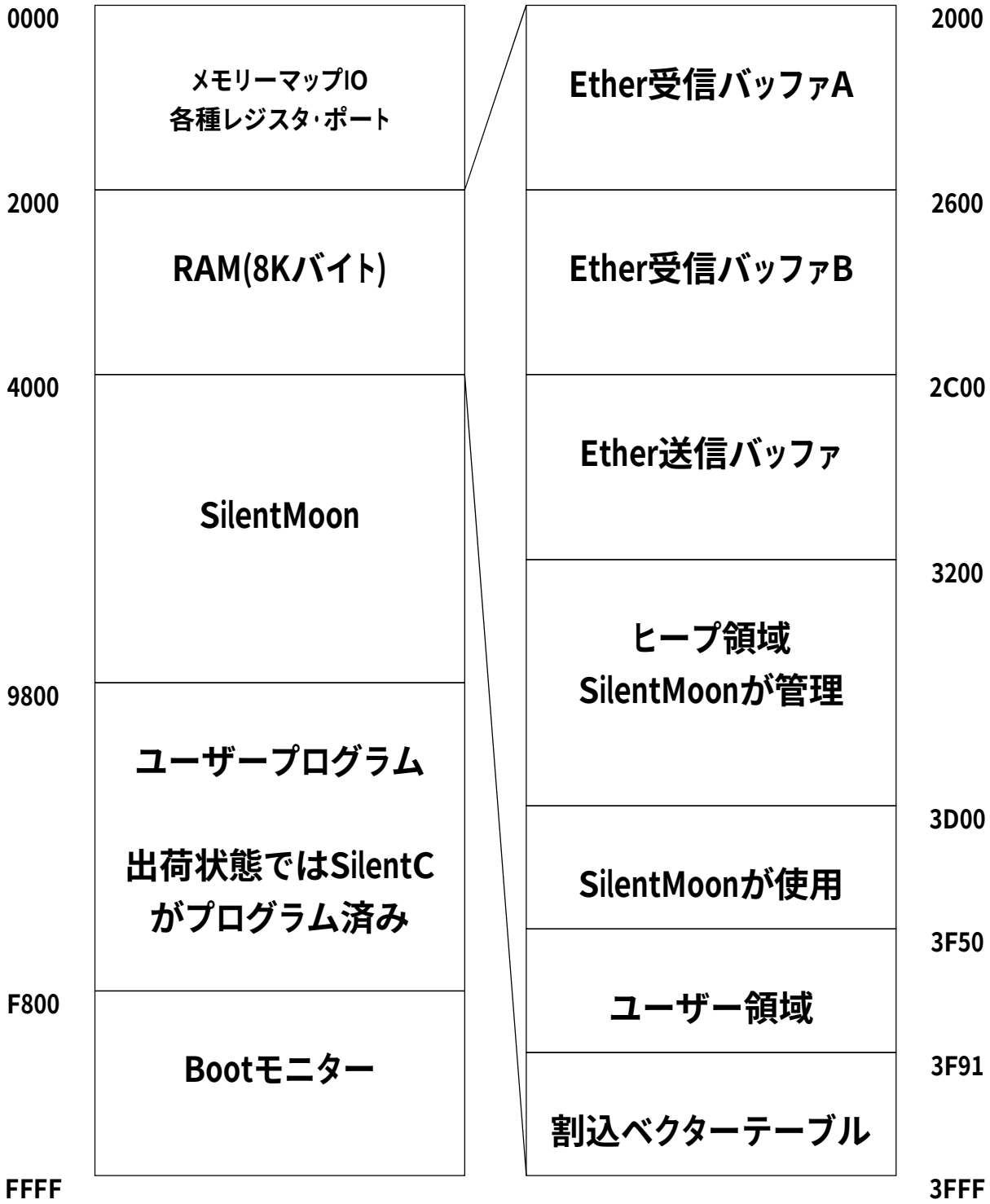
SilentMoon上で動作するユーザーが作成したプログラムを配置するブロックです。無料でダウンロードできる統合開発環境であるCodeWarrior Development Studioを利用してユーザーが開発したプログラムを実行できます。

工場出荷状態のＯＳ－１のユーザープログラムエリアにはSilentCと呼ばれる会話型Ｃ言語インタープリターが書き込まれています。殆どのユーザーアプリケーションはSilentCを用いて簡単に開発できます。またより高速性やリアルタイム性を求めるアプリケーションを動作させる場合にはSilentCを消去して代わりにユーザーのプログラムを書き込む事も可能です。

作成したプログラムファイルをＯＳ－１に転送するだけでリセット時にユーザープログラムエリアを消去してからプログラムファイルを読み込んでフラッシュメモリーを自己プログラムします。工場出荷状態に復帰すればSilentCはいつでも復活します。

メモリーマップ

ＯＳ－１のメモリーマップは以下の通りです。



外部端子と予約ポート

ＯＳ－１の外部端子はMC9S12NE64の殆どの外部端子をそのまま取り出してあります。
MC9S12NE64の端子に関してはFreeScaleのマニュアル(英語)をご覧下さい。
ＯＳ－１とＯＳIＯ－１の組み合わせで利用可能なポートは以下の通りです。

・PortG0からPortG4まで
・PortJ0からPortJ3まで
・PortL0
・PortT5からPortT7まで
・A/D入力ポート8chすべて
・SCIの1チャンネル目

また以下のポートはＯＳIＯ－１が使用していますがＯＳ－１を単体で使用する場合には利用可能になります。必ずポートの入出力の方向を再設定してからご利用下さい。

・PortH0からPortH3はSW1からSW4に割り当てられています。SW1はリカバリスイッチなのでリセット時にはHIGHである必要があります。LOWに落としたままリセットすると永久にリカバリを繰り返しますのでご注意下さい。
・PortH4からPortH6はLED1からLED3に、PortJ6はLED4に割り当てられています。
・PortJ7はリレー1に、PortT4はリレー2に割り当てられています。
・PortG5はシリアルのRTSにPortG6はCTSに割り当てられています。
・SCIの0チャンネル目はシリアルポートに割り当てられています。

上記のポートのうちPortH4からPortH6、PortJ6とPortJ7、PortT4、PortG5はリセット時にポートの方向が出力にプログラムされますので接続するデバイスの方向に十分注意して下さい。出力ポートを出力ポートに接続すると大きな電流が流れて半導体を破損する可能性があります。

外部端子に関する注意事項

外部ポートに信号を接続する場合には25mA以上の電流は取り出せませんし25mAを超えて流し込めません。またＯＳ－１の内部の電圧(VDD)は3.3Vです。5VのTTLレベルを直接接続できませんので注意して下さい。

また印加する電圧はVDDより高電圧にならないようにする必要があります。また複数のポートに連続的に電流を流す場合にはチップの温度上昇に留意して下さい。以上の注意事項を守らないと最悪の場合ＯＳ－１の内部を破損する可能性があります。

SilentMoonオペレーティングシステム

SilentMoonは22Kバイトとサイズが小さいながらも組み込み機器に求められる必要かつ十分なサービスを提供するマルチスレッドＯＳです。

システムコールを呼び出すことでマルチスレッド機能、TCP/IP、UDPなどのネットワークスタック、ファイルシステム、メモリ管理などのサービスを提供します。組み込み機器に求められるサービスの殆どを実装していますのでユーザーが容易に専用プログラムを作成することが出来ます。

SilentMoonは10ms間隔のタイマー割り込みによって現在実行中のスレッドを抜け出して次のスレッドに移動するプリエンティブなマルチスレッド機能を搭載しています。スレッドの実行優先順位の設定は実装されていませんのでラウンドロビン方式で登録されているスレッドを順番に実行します。仮にスレッドが無限ループを繰り返している場合でも割り込みによりタスクスイッチが可能です。

こうしたマルチスレッド機能を利用してSilentMoonの起動時にシステムスレッドとユーザースレッドの二つのスレッドを開始します。システムスレッドはもっぱらネットワークのトランスポート層のステート監視を行いTCP/IPスタックを提供します。ユーザースレッドはユーザープログラムを実行します。工場出荷段階で書き込まれているSilentCはこのユーザースレッドとして動作する事になります。

面倒なTCP/IPのセッション管理が完全にユーザーから隠蔽されていますのでソケットライブラリを利用してサーバーとクライアントの両方のプログラムが容易に可能です。

SilentMoonの機能

・カーネル

OSの心臓部でスレッド管理、メモリー管理、ファイル管理、レジストリ管理を担当しています。

・ネットワークスタック

カーネルが提供するマルチスレッド機能とメモリー管理機能を利用してUDPとTCP/IPのプロトコルスタックが実装されています。ユーザーはソケットインターフェースを利用してネットワークにアクセスできます。

・HTTPサーバー

SilentMoonの持つソケットサービスを利用したHTTPサーバーが実装されています。このHTTPサーバーはCGI機能を持っていますのでWebページを利用してＯＳ－１のすべてのポートの監視や設定が可能です。またユーザの製作したプログラムにWebページを通じてアクセスできる機能を提供しています。

・TFTPサーバー

ネットワーク経由で外部のコンピュータとファイルを交換するためにTFTPサーバーを内蔵しています。Windowsなどに標準で付属しているTFTPクライアントを利用してＯＳ－１とWindowsパソコンの間でファイル交換が可能です。またサイレント製のファイル交換ソフトである「デバッグ文鳥」を利用するとドラッグアンドドロップを利用して直感的なファイル交換が可能です。

・DHCPクライアント＆サーバー機能

LANにあるDHCPサーバーからIPアドレスをはじめとするネットワーク設定の割り当てを受けて自動的に設定するDHCPクライアント機能を実装しています。またＯＳ－１とパソコンを直接接続する際にはＯＳ－１がDHCPサーバーとして動作してパソコン側のＬＡＮを自動的に設定します。割り当てるIPアドレスはＯＳ－１に設定されているIPアドレスの次のIPアドレスに固定されています。

・セルフプログラムマネージャー

起動時に特定のファイルが存在していればそのファイルの内容を自分自身のフラッシュメモリに自己プログラムする機能を提供しています。組み込んでしまった機器のプログラムをネット経由で書き換える事が可能になります。

・豊富なサンプルプログラム

ユーザーは24kバイトのユーザープログラムエリアに自由にプログラムを書き込めます。サイレントはこのユーザーアプリケーションとして様々なサンプルプログラムを用意しています。また他のＯＳ－1のユーザーから公開を許されたサンプルも随時公開していきます。以下はその一例です。

・SilentC

インタープリター型のＣ言語で出荷状態のＯＳ－1に書き込まれているアプリケーションです。

・Telnetサーバー及びクライアント

Telnetプロトコルを利用してシリアルポートをアクセスするプログラムです。ネットワークを通じてシリアルポートをアクセスしたり2台のＯＳ－1を組み合わせてシリアルデータをTCP/IPを利用して交換する事ができるサンプルです。このサンプルはソースも公開されています。

・POPクライアント及びSMTPクライアント

ＯＳ－1がメールを受け取りメールを返信してくれるプログラムです。この機能を拡張すれば携帯電話などで外部からＯＳ－1のポートを操作する機能を実現できます。ソースも公開されます。

SilentMoonのファイルシステム

ＯＳ－１には1メガバイトのシリアルフラッシュを搭載し、このデバイス上にファイルシステムを構築しています。1メガバイトの容量のうち128kバイトは工場出荷状態へ復帰するためのデータとデフラグに必要な作業領域として予約されていますので実質的には896kバイトの容量を利用できます。

フラッシュメモリーの特徴として消去するとメモリーの値はすべて1となります。一度すべてのデータを1にした後に指定したビットだけ0にする事ができます。これがフラッシュメモリーの書込み動作です。逆に言えばフラッシュメモリーは1を0に書き換える事は可能ですが0を1にする事はできないと言う事です。もしどうしても0を1に書き換えたい場合には一度フラッシュをすべて消去してから改めて必要なビットだけを0にプログラムするという動作が必要になります。

こうした特長を持つフラッシュメモリーにファイルシステムを搭載するので通常のハードディスク上などのファイルシステムと比較すると以下のような制限があります。

・同時に書き込めるファイルの数は1個だけ

SilentMoonはマルチスレッドOSですから複数のプログラムから同時にファイルを書く事態が発生する可能性があります。しかしフラッシュメモリの構造上は複数のアドレスを同時にプログラムする事ができませんので後からファイルを書こうとしたスレッドはファイルを書き込めません。

・ファイルを削除してもファイルシステム内に無効な領域が残る

通常のファイルシステムであればファイルを削除するとその領域は完全に削除されて空き容量も増加しますが、ＯＳ－１のファイルシステムではファイルを削除するとファイルの先頭に削除されたというマークが書き込まれるだけなので無効な領域として残ります。またファイルの名前を変更する場合も同様に削除マークが書き込まれて新しいファイル名からのシンボリックリンクが張られます。この無効領域を整理するのがデフラグです。

・ファイルを書き込み中に電源を切るとすべて無効な領域になる

SilentMoonはファイルを書き込む際にはまず仮のマークを書き込みます。書込みがすべて成功したらその時点で通常ファイルであるマークを書き込みますが、このマークを書き込む前に電源が切られるとそれまで書き込んだ領域は無効な領域として残ります。

デフラグ

無効な領域が増加してくると必要なファイルを書き込む領域が不足してしまいますので、その際にはデフラグを実行します。これはファイルシステムの先頭から順番にファイルを読み込んで有効なファイルだけを予約領域にコピーしていき予約領域がいっぱいになればその内容をファイルシステム側に書き戻すという作業を繰り返す事でファイルシステム内の無効なファイル領域を一掃する操作です。

デフラグはフラッシュメモリーをファイルとして利用する以上は必ず考慮しなければならない処理ですので概念と必要性だけはご理解頂けたら幸いです。

デフラグにはかなりの時間がかかる場合があります。ファイルの数が多い場合には最悪で数分程度の時間がかかります。デフラグの最中は絶対に電源を切らないで下さい。万が一デフラグの最中に電源が切れるとファイルシステム全体が復旧不能になる恐れがありますのでご注意ください。こうなってしまうと工場出荷状態に戻さなければそれ以降ファイルシステムが利用できなくなってしまいます。

こうした事態に備えて予めパソコンにＯＳ－1からファイルを転送してバックアップをとっておく事を強くお勧めします。

・フラッシュの書込み回数制限について

ＯＳ－1に採用しているシリアルフラッシュの標準の書き込み回数の上限は１０万回です。毎日１０回ずつ同じ領域に書き込んだ場合には２７年程度の寿命があります。基本的にファイルを書き込む際にはどんどんと書き込む領域を移動させながらに書いていきますので、同じ領域に書き込む頻度はかなり小さくなります。

デフラグを行うとまたフラッシュの先頭から領域を使う事になりますので、実質的にデフラグの回数が１０万回までは問題なく使用できるわけです。厳密に言えばこまめにデフラグをするよりある程度無効領域が大きくなってからデフラグをした方がフラッシュの書込み回数の上限という観点からは望ましいと言えますが１０万回という回数の大きさを考えるとまり気にする必要はないものと思います。

HTTPサーバー

SilentMoonにはWebブラウザを利用してＯＳ－１を操作するためのHTTPサーバーが実装されています。ブラウザ経由でＯＳ－１のすべてのポートをコントロールできます。またユーザーが用意した関数を実行したりユーザーのメモリーをブラウザで操作できます。ユーザーが自分で作成したhtmlファイルをＯＳ－１に転送する事で自由にWebページを変更する事も可能です。

・SilentMoonの標準設定ページについて

ＯＳ－１の工場出荷状態のWebページは以下の設定が可能です。すべての設定はリセット後に有効になりますので設定後に必ずリセットしてください。

・IPアドレス設定

ＯＳ－１が使用するネットワークを設定します。IPアドレスをDHCPで取得する場合にはどのIPアドレスがＯＳ－１に割り当てられたか把握する必要があります。DNSのIPは通常はルーターのIPで構いません。

・イーサネット設定

Ethernetの物理的な能力の設定を行います。現在の相手方の殆どが100Mbpsの全二重で通信できますので出荷時の設定で問題ありません。しかし一部の機器で全二重を認識しない場合もありますので、この場合には自動設定をお試しください。

・サービス設定

起動時にDHCPサーバーやTFTPサーバーを有効にするかどうかの設定を行います。またパスワード欄にはＯＳ－１にアクセスする際のパスワードを設定できます。TFTPでファイル転送をする前にパスワードと同じファイル名をgetするとそれ以後のTFTPが有効になります。またHTTPサーバーにアクセスする際のWebログインは3分間有効です。それぞれのサービスはRAMを128バイト消費しますので不要であれば無効にする事をお勧めします。

・SilentC設定

telenet経由でSilentCにアクセスする際の設定を行います。またシリアルポートの設定も可能です。シリアルポートを無効にする事でSilentCからシリアルポートをフルコントロールできるようになります。ＯSIＯ－１にシリアル機器を接続してSilentCでコントロールしたい場合には無効にして下さい。この場合はtelnetでプログラミングを行うことになりますのでtelnetを有効にしておかなければSilentCにアクセスできなくなりますのでご注意ください。シリアルでSilentCにアクセスできる場合にはtelnetを無効にする事でRAM容量が128バイト開放されます。

CGIについて

SilentMoonのHTTPサーバーにはCGI機能が搭載されています。CGI機能を利用するとWebページからポートやメモリーを操作できます。またユーザープログラムをWebページからの指示で実行することも可能です。CGIを利用してＯＳ－１を操作するにはHTMLで記述したファイルを準備します。すでにＯＳ－１の内部にはいくつかの.htmという拡張子のついたファイルがありますので中を覗いて見てください。拡張子が.htmというファイルのみCGI処理が行われます。同じHTMLファイルであっても.htmlという4文字の拡張子のファイルはCGI処理が行われません。またそれ以外のファイルタイプとして.gifと.jpgが利用可能です。詳しくはＯＳ－１プログラミングマニュアルを参照してください。

・CGIの使い方

CGI機能とはHTML文中に埋め込んだ特定の文字列を置換する機能です。HTML文中で$あるいは#に続けてシンボル名を記述するとHTTPサーバーはCGIシンボルとして文字列の置換処理を行います。たとえば$IPという文字列がHTML文中に出現すると$IPは現在のＯＳ－１のIPアドレスを表す文字列に置換されます。ここで注意して頂きたいのはIPアドレスは最大で１５バイトもの長さになってしまうという事です。このため$IPというシンボルの先頭から15文字分は予めスペースを確保しておくと言う事が重要です。#はビット反転になります。$IPという文字列を置換せずにそのまま表示したい場合には.htmではなく.htmlという拡張子のファイルを利用します。

また逆にIPアドレスを設定したい場合にはHTMLのFORM機能を利用してIP=192.168.1.10という文字列を任意のhtmファイル名に?を付加して記述すれば代入できます。FORM内の複数の代入文は&で区切って記述できます。システムで予約されているCGIシンボルはOS-1プログラミングマニュアルをご覧下さい。

・ポート操作用のサンプルHTMLファイル

以下の内容のファイルをTest.htmという名前で作成してＯＳ－１に転送してWebブラウザからhttp://192.168.1.10/Test.htmでアクセスして下さい。ＯＳ－１のボード上にあるLEDをWebブラウザから操作できます。

```
<HTML><HEAD></HEAD><BODY><CENTER><FORM action="Test.htm">
<INPUT type="radio" name="PCHKL3" value=0 #PCHKL3> ON
<INPUT type="radio" name="PCHKL3" value=1 $PCHKL3> OFF
<INPUT type="submit" value="設定">
</FORM></CENTER></BODY></HTML>
```

TFTPサーバー

SilentMoonにはパソコンとファイルを交換するためのtftpサーバーが実装されています。設定ページでtftpサーバーを有効にするとSilentMoonの起動時にtftpサーバーのスレッドを開始します。パソコンとＯＳ－１のファイルをやりとりしたい場合にはWindowsに標準で付属しているtftpクライアントを利用する事になります。

・ＯＳ－１のtftpサーバーの拡張機能

ＯＳ－１のtftpサーバーには独自の拡張機能を実装しています。通常tftpプロトコルはUDPを利用して最低限のファイル交換を行うプロトコルですがＯＳ－１のtftpサーバーは以下の予約されているキーワードをファイル名としてＯＳ－１に要求すると特別な動作をするように設計されています。Windowsのコマンドプロンプトを起動して以下のようにtftpコマンド打ち込みます。

tftp 192.168.1.10 get dirlist ‥‥ ＯＳ－１側のファイルの一覧が表示されます。

tftp 192.168.1.10 get DefragFileSystem! ‥‥ ＯＳ－１のファイルシステムを最適化します。最適化中はＯＳ－１のボード上のLEDが点灯します。最適化中は絶対にリセットをしたり電源を切らないで下さい。終了するとDoneを返します。

tftp 192.168.1.10 get DeleteFilefilename ‥‥ filenameというファイルを削除します。ターゲットとなるファイル名はDeleteFileに続けてスペースを空けずに記述します。成功するとDoneを返します。

・tftpによるファイル転送

tftpを利用してファイルを転送するときにはWindowsのコマンドプロンプトを起動してtftpコマンドを使います。tftp IPアドレス get ファイル名で吸出し、tftp IPアドレス put ファイル名で送り込みになります。バイナリファイルをやりとりする場合にはtftpの後に-iを付加します。
ＯＳ－１からパソコンへファイルを吸い出す場合にはtftpのgetコマンドを使用します。
tftp 192.168.1.10 get filename ‥‥ テキストファイルの場合
tftp -i 192.168.1.10 get filename ‥‥ バイナリファイルの場合

パソコンからＯＳ－１へファイルを送り込む場合にはtftpのputコマンドを使用します。
tftp 192.168.1.10 put filename ‥‥ テキストファイルの場合
tftp -i 192.168.1.10 put filename ‥‥ バイナリファイルの場合
ＯＳ－１側に同名のファイルが存在していた場合には失敗します。まず削除して下さい。

SilentMoonのシステムコール

この章ではSilentMoonで提供されるシステムコールに関して解説します。またSilentMoonのシステムコールの詳細に関しては別冊のＯＳ－１プログラミングマニュアルを参照してください。
UINT8はunsigned charのUINT16はunsigned shortの別名のtypedefです。

・スレッド管理機能

SilentMoonは10ms間隔のタイマーにより複数のプログラムの実行を切り替えるプリエンティブなマルチスレッド機能を提供しています。この機能はネットワークのサービスには不可欠でユーザープログラムの裏側でネットワーク処理を行うために実装されています。関連のシステムコールは以下の通りです。

・UINT8 CreateThread(UINT16 adrs, UINT16 stacksize, UINT16 arg);

新しくスレッドを生成します。実行するスレッドのスタートアドレスとスタックのサイズと引数を与えます。成功するとスレッドハンドルを返します。

・void KillThread(UINT8 handle);

スレッドハンドル指定してスレッドを終了させます。0を与えると自分自身のスレッドが終了します。スレッドの生成時にシステムに要求したスタックエリアは開放されます。

・void SystemSleep(void);

システムに制御を戻します。これにより自分のスレッドに残っている実行時間を他のスレッドが利用できるようになります。イベント待ちうけのループ内で毎回呼び出す事でシステム全体のパフォーマンスが向上します。

・void WaitCriticalSession(UINT8 *mutex);

ミューテックスを取得してクリティカルセッションの同期をとります。ミューテックスのアドレスを指定します。デバイスなどを複数のスレッドでアクセスする際の排他管理に利用すると便利です。

・void Sleep(UINT16 time);

指定時間の間スレッドの実行を休止します。10ms単位で時間を指定します。空ループなどで時間を費やすとシステム全体のパフォーマンスが低下するので必ずSleepを利用して時間を調整してください。

・メモリー管理機能

SilentMoonにはRAMを有効に活用するためにメモリー管理サービスを提供しています。基本的にはメモリーが極めて小さいシステムなのでユーザーはメモリー管理機能を利用してメモリーを確保して不要になれば直ちに開放する必要があります。

・void *MemoryAlloc(UINT16 size);

ヒープメモリーを割り当てます。割り当てに失敗するとNULLポインタを返します。

・void MemoryFree(void *);

割り当てられたメモリーを解放します。指定するポインタはMemoryAllocで割り当てられたものでなければなりません。もし0を指定すると何もしません。

・タイマー機能

ネットワークのタイムアウトなどを処理するためタイマー機能を提供しています。タイマーの分解能は最小で10msです。特にCreateTimerには多くのオプションがあり基本的なカウンター動作以外にも繰り返し動作や関数コールバックや自動スレッド生成などの豊富な機能があります。

・UINT8 CreateTimer(UINT8 mode, CallbackFunc8 func, UINT16 val, UINT8 timer, UINT8 stack);

タイマーを生成及び開放します。modeには以下のタイマーのアクションを指定します。

0:何もしない　1:シンプルなカウントダウン動作　2:指定時間後に関数呼び出してタイマーを開放
3:指定時間後に関数を呼び出す 4:指定時間後にスレッドを生成しタイマーを開放
5:指定時間後にスレッドを生成する

funcには呼び出される関数やスレッドのアドレスを、valには10ms単位でのカウンター値を、
timerにはタイマーハンドルを、stackにはスレッド生成に必要なスタックサイズを指定します。
タイマーハンドルに0を指定した場合には空いているタイマーを割り当ててそのタイマーハンドルを返します。0以外を指定した場合には指定されたハンドルのタイマーを再設定します。この際にvalを0にするとタイマーを開放します。

・UINT16 GetTimerCount(UINT8 timer);

指定されたハンドルのタイマーのカウンター値を得ます。カウントダウンは0でストップします。再度カウンタをセットする場合にはタイマーハンドルを指定してCreateTimerを呼び出して下さい。

・ファイルシステム機能

ＯＳ－１には1メガバイトのシリアルフラッシュを搭載し、このデバイス上にファイルシステムを構築しています。ファイルを操作するための各種システムコールが用意されています。ファイルハンドルの0は書込み用ハンドルとして固定されています。書き込みモードでオープン中でも読み込みモードでオープン可能です。また読み込みモードでオープンする数には制限がありません。メモリーが確保できる限りオープンが可能です。

・void DefragFilesys(UINT16 key);

ファイルシステムの無効領域を整理して先頭からファイルを再配置しなおします。最悪の場合には数分程度の時間が必要です。不用意な実行を防ぐためkeyには16787を指定します。

・UINT32 GetFinalPos(void);

ファイルシステムの次にファイルを書くポインタを求めます。917500が物理的なリミットですので減算すればファイルシステムの残りサイズを求められます。

・UINT8 CreateFile(INT8 *filename);

ファイルを書き込みモードでオープンします。同時に書き込みモードでオープンできるファイルは一つだけですので、すでに書込みモードでオープンしている場合には失敗します。またすでに同名のファイルが存在している場合にも失敗します。戻り値は以下の通りです。

0:成功　1:すでに書き込みモードでオープンされている　2:ファイル名が無効　3:すでに指定のファイルが存在している　4:ファイルシステムの容量が足りない　5:割り当てるメモリーが不足

・UINT16 WriteFile(void *ptr, UINT16 size);

ファイルにデータを書き込みます。書き込むべきデータのポインタとサイズを指定します。実際にファイルに書き込まれたサイズを返します。書込みエラーが発生した場合には0を返します。

・void CloseFile(UINT16 handle);

ファイルをクローズします。書込みモードでオープンしているファイルをクローズする場合には0を指定します。読み込みモードでオープンされているファイルをクローズする場合にはファイルハンドルを指定します。ファイルをクローズしないと確保したメモリーが解放されませんので読み込みモードであっても必ずファイルをクローズしてください。

・UINT8 DeleteFile(INT8 *filename);
指定したファイル名のファイルを削除します。成功した場合には0を、ファイルが見つからなかった場合には1を返します。

・UINT8 MoveFile(INT8 *filename, INT8 *newfile);
ファイルの名前をfilenameからnewfileへ変更します。戻り値は以下の通りです。

0:成功　1:ファイルが見つからない　3:すでに指定のファイルが存在している　4:ファイルシステムの容量が足りない　5:割り当てるメモリーが不足

・INT8 *FindFile(UINT8 init);
ファイルシステム内に存在しているファイル名を順番に得ます。最初にinitに1を指定して呼び出します。二回目以降はinitを0にして呼び出します。戻り値はファイル名へのポインターです。ここで大切な事はこのポインターはFindFileの内部でMemoryAllocによって確保された領域のポインタですので必ずユーザー側で解放して下さい。それ以上ファイルが存在しない場合にはNULLポインタを返します。

・UINT32 GetFileSize(INT8 *filename);
ファイルのサイズを得ます。もし指定されたファイルが存在しない場合には0を返します。

・UINT16 OpenFile(INT8 *filename);
ファイルを読み込みモードでオープンします。成功すればファイルハンドルを返します。ファイルが見つからなかった場合には0を返します。オープンしたファイルは必ずクローズして下さい。

・UINT16 ReadFile(UINT16 handle, UINT8 *buf, UINT16 size);
ファイルからデータを読み込みます。ファイルハンドルと読み込むバッファのポインタとバッファのサイズを指定します。実際に読み込んだサイズを返します。

・UINT32 SeekFile(UINT16 handle, UINT32 pos, UINT8 org);
ファイルの読み込みポインタの位置を調べたり変更したりします。handleにはファイルハンドルを指定します。orgにはどこを基準に読み込みポインタを設定するか以下の値を指定します。

0:ファイルの先頭からの位置を指定する　1:現在の読み込み位置を相対的に変更する
2:ファイルの終端からの位置を指定する

posには位置を指定します。SeekFileは現在の読み込み位置を返します。これを利用してposに0をorgに1を指定すると現在のファイルの読み込み位置を知る事が出来ます。

・ネットワーク機能

ＯＳ－１にはネットワークを利用するための豊富なシステムコールが準備されています。使い方は一般的なソケットプログラミングに対応していますのでわずかな改造でその他のプラットフォームで稼動しているアプリケーションを移植できるように考慮されています。詳しいネットワークプログラミングについてはＯＳ－１プログラミングマニュアルをご覧下さい。

・INT8 CreateSocket(UINT8 prot);

ソケットを生成します。protに0を指定するとUDPソケットが、1を指定するとTCP/IPソケットが生成さます。またprotに2以上の数値を設定するとprot*10秒のアライブタイマーが設定されます。アライブタイマーで設定された時間内に全くデータのやり取りがなかった場合には強制的にセッションを切断します。最短のアライブタイマーは20秒と言うことになります。成功すればソケットハンドルを返します。

・void CloseSocket(INT8 sochandle);

ソケットを開放します。ソケットハンドルを指定します。ソケットを開放する事で内部的に確保されていたメモリーが解放されますので生成したソケットは必ず解放して下さい。

・UINT8 Bind(INT8 sochandle, UINT16 port, UINT8 maxsoc);

ソケットにポート番号を関連付けます。ソケットハンドルを指定してportには「自分の」ポート番号を指定します。maxsocにはTCP/IPでAcceptを利用して先方からの接続を待ち受ける際に同時にいくつまで新しい接続を受け付けるかを指定します。Acceptを利用してサーバーモードで動作させる場合には1以上の数を指定しないと接続を受け付けません。成功すると1を返します。

・INT8 Connect(INT8 sochandle, UINT32 ip, UINT16 port);

TCP/IPで相手側と接続を確立します。ソケットハンドルと相手側のIPアドレスと相手側のポート番号を指定します。タイムアウトは10秒です。戻り値は以下の通りです。

1:成功　-1:ソケットが存在しない -2:タイムアウト発生(10s) -3:ステートエラー

・INT8 Accept(INT8 sochandle, UINT16 timeout);

サーバーモードで相手からのTCP/IP接続を待ちます。Bind済みのルートソケットハンドルを指定します。timeoutには10ms単位のタイムアウト時間を指定します。timeoutに0を指定するといつまでも接続を待ち続けます。相手からの接続があった場合には新たにソケットを生成してそのソケットハンドル(正の値)を返します。その他の戻り値は以下の通りです。

-1:ソケットが存在しない -2:タイムアウト発生 -3:ステートエラー

・INT16 Read(INT8 sochandle, UINT16 timeout);
TCP/IPで相手からデータが到着したかどうか調べます。有効なTCP/IPソケットハンドルと10ms単位のタイムアウト値を指定します。timeoutに0を指定した場合には呼び出した時点でデータが到着していなければ直ちに-2を返します。データを受信していた場合にはデータの長さを返します。それ以外の戻り値は以下の通りです。

-1:ソケットが存在しない -2:タイムアウト発生 -3:ステートエラー

・UINT8 *GetReceiveBuffer(INT8 sochandle, UINT8 release);

受信したデータが格納されているバッファのポインタを返します。UDPとTCP/IPの両方に共通して利用します。必ずReadやRecvFromでデータが到着した事を確認してからこの関数を呼び出して下さい。またこの関数で返されるデータのポインタは内部的にMemoryAllocで確保されたものですのでデータの処理が終わった後には必ずMemoryFreeで開放して下さい。開放を忘れるとあっという間に受信データでRAMが一杯になってしまいます。またreleaseに1を指定すると受信したデータのバッファをソケットから切り離して新たなるデータを受信できる状態になります。
TCP/IPであればこの時点で相手にACKが送信されます。通常はreleaseには1を指定して呼び出して下さい。releaseに0を指定すると受信データが残ったままになりますので結果的にこの関数を呼び出す度に同じポインタが返されます。

・INT16 Write(INT8 sochandle, UINT8 *buf, UINT16 len);
TCP/IPで相手にデータを送信します。有効なTCP/IPソケットハンドルと送信するデータのポインタと長さを指定します。戻り値は送信したデータの長さ(正の値)を返します。それ以外の戻り値は以下の通りです。

-1:ソケットが存在しない -3:ステートエラー

・INT8 CheckWriteComplete(INT8 sochandle);
TCP/IPの送信が終了したかどうか確認します。1ならTCP/IPでデータが無事に届いた事を示します。0であればまだ先方には届いていません。それ以外の戻り値は以下の通りです。

-1:ソケットが存在しない -2:タイムアウト発生 -3:ステートエラー

・INT8 WaitWriteComplete(INT8 sochandle);
TCP/IPの送信が終わるまで待ちます。戻り値は上記のCheckWriteCompleteと同じです。
送信が完了するまで待つ間は内部的にSystemSleepを呼び出して他のスレッドが有効にCPUタイムを利用できます。

・INT16 SendTo(INT8 sochandle, UINT32 ip, UINT16 port, UINT8 *buf, UINT16 len);
UDPで相手にデータを送信します。UDPソケットハンドル、相手方のIPアドレスとポート番号、送信するデータのポインタと長さを指定します。戻り値は送信したデータの長さを返します。ソケットが存在しない場合には-1を返します。UDPの送信は出しっぱなしですので待ちません。

・INT16 RecvFrom(INT8 sochandle, UINT16 timeout);
UDPで相手からデータを受け取ります。UDPソケットハンドルと10ms単位のタイムアウト値を指定します。もしUDPデータが到着したらその長さを返します。それ以外の戻り値は以下の通りです。

-1:ソケットが存在しない -2:タイムアウト発生

もし有効なデータが到着したら早速releaseに1を指定してGetRecieveBufferを呼び出し、受信データバッファのポインタを得てソケットから切り離して下さい。この操作をしなければその後に到着するUDPデータは受信出来ずに捨てられてしまいます。

・UINT32 GetSenderIP(INT8 sochandle);
TCP/IPおよびUDP通信の相手先のIPアドレスを得ます。有効な受信データが到着した後に呼び出して下さい。

・UINT16 GetSenderPort(INT8 sochandle);
TCP/IPおよびUDP通信の相手先のポート番号を得ます。有効な受信データが到着した後に呼び出して下さい。

・INT16 GetDefMtu(INT8 sochandle, UINT16 mtu);
TCP/IP通信の際のパケットの大きさを指定します。有効なTCP/IPソケットハンドルを指定します。mtuを0にして呼び出すと現在設定されているMTUを返します。それ以外であればmtuをそのまま返します。SilentCが動作中には512を指定するようにして下さい。

・UINT32 GetHostByName(UINT8 *name);
nameで指定されたドメイン名を元にIPアドレスを求めます。成功すれば32ビットのIPアドレスを返します。失敗した場合には0を返します。

・レジストリ機能

ＯＳ－１が起動する際にファイルから設定値を読み込んでメモリーにセットする機能を提供します。レジストリはテキストファイルですので簡単に内容を確認・変更できます。

・void LoadRegistry(VarEntry* RegTable);

レジストリを読み込みます。レジストリのファイル名や各シンボルの名前や属性などはすべてRegTable内に記述します。RegTableへの記述方法はＯＳ－１プログラミングマニュアルを参照して下さい。

・INT8 SaveRegistry(VarEntry* RegTable);

レジストリを保存します。RegTableの指定に従ってテキストファイルを出力します。

・void UpdateString(VarEntry* RegTable);

レジストリで読み込んだ文字列はヒープ領域に用意されます。ユーザーがこの文字列ポインタに別の文字列を代入した後にこの関数を呼び出します。古い文字列の領域を開放して新しい文字列の領域を新たにヒープ領域に確保します。

・その他のシステムコール

その他にシリアルポート関連のシステムコールやシリアルポートを利用した各種のデバッグ出力またデバイスの初期化などのシステムコールが存在しますがここでは説明を省略します。個々のシステムコールに関しては別冊のＯＳ－１プログラミングマニュアルを参照してください。

インタープリター型C言語　SilentC

SilentCはインタープリタ型のC言語です。C言語に準拠した文法でプログラムが可能です。シリアルターミナルあるいはtelnetでSilentCにアクセスするだけでプログラム開発とデバッグが完結します。殆どの簡単なアプリケーションであればSilentCを利用して短時間にプログラムを完成できます。またSilentCには以下の特徴があります。

・Cコンパイラが不要です

SilentCはインタープリター言語です。ソースを修正すればすぐにその結果が反映します。コンパイルやリンクやターゲットへの転送の手間が省略されます。組み込んでしまった機器のデバッグや現場での仕様変更にも大変有効です。

・ダイレクト実行が可能です

SilentCで書かれたプログラムはいつでも中断ができます。またコンソールからコマンドを入力すると変数の値を確認したり任意の関数を実行してその戻り値を確認したり出来ます。Cのステートメントを直接入力する事でポートを操作したり関数をダイレクト実行できます。ちょうどVisualBasicや昔のマイコンに内蔵されていたBASICと同じ感覚です。

・ファイル内の関数をダイレクトに呼び出せます

SilentCのプログラムはRAMではなくファイルに置かれています。これを利用してファイルの中の関数を呼び出す事ができるので一度作成したSilentCプログラムを自分だけのライブラリとして再利用できます。

・ユーザードライバとの組み合わせが可能です

より高速性やリアルタイム性が求められるアプリケーションに対応するために1kバイトまでのユーザードライバとのリンクが可能です。ユーザーが割り込みハンドラなどを用意してSilentCと連携させる事でより広範囲のアプリケーションに対応できます。

SilentCのコマンドプロセッサー

コマンドプロセッサーはWindowsのコマンドプロンプトやUnixのシェルに相当するコマンドインタープリターです。ユーザーが打ち込んだコマンドを解釈してそれを実行します。SilentCのプログラムを作成したりする際のユーザーとの接点となりますので、まずはこのコマンドプロセッサーの機能を十分に把握してください。

・再入力機能

Enterを押した後の最初のキー入力が上矢印キーであった場合にのみ直前の入力を編集できます。この機能は頻繁に使う機能ですので確実に確認して下さい。またESCキーを二度押すと入力している内容を破棄してなにも入力しないでEnterだけを押した事と等価になります。

・ファイルコマンド

ファイルを操作するためのコマンド群です。

・dir

ファイル名とファイルサイズのリストを表示します。またファイルの無効領域の大きさも表示します。

・type ファイル名

ファイルの内容を表示します。

・delete ファイル名

指定したファイルを削除します。それまでそのファイルが使用していたデータ領域は無効領域としてファイルシステム内に居座ります。

・defrag

ファイルの無効領域を整理します。実行中はＯＳ－１の基板上のLEDが点灯します。数分かかる場合があります。またデフラグ中に電源を切るとファイルシステムが破壊されます。その場合は工場出荷に戻す操作を行って下さい。

・編集コマンド

プログラムを編集するためのコマンド群です。昔のBASICの編集方法に近いです。saveするまではいかなるファイルの内容も変更されません。

・load ファイル名

指定したファイルを編集モードにします。ファイル名を省略すると直前のファイル名が用いられます。ファイルが読み込む際に10から始まって10ずつ増加する行番号が付加されます。10ずつ増加しているのは行と行の間にテキストを挿入できるようにしているためです。

・save ファイル名

指定したファイルに現在編集している内容を書き込みます。ファイル名を省略すると直前のファイル名が用いられます。この場合にはファイル名の確認を求められますのでよろしければyを答えて下さい。読み込み時に付加された編集用の行番号は書き込みの際にすべて取り除かれます。

・list 開始行番号-終了行番号

指定された行番号の範囲を表示します。行番号を省略するとすべての行を表示します。list -100は先頭から100まで、list 300-は300から終わりまでを表示します。

・edit 行番号

指定された行番号をカーソル編集モードにします。左右カーソルとBackSpaceとDeleteが使用可能です。ハイパーターミナルなどの一部の端末でDeleteはCTRL+BSでも代用可能です。ESCを二度押すと入力を中止できます。editを利用して指定の行を表示させて行番号部分だけを書き換えると同じ行をコピーできます。コピーしておいてからそれぞれのテキストを削除する方が編集が楽になります。Enterを押すと変更を確定します。

・new

何も入力されていない初期状態になります。新規にファイルを作成する場合にはまずこのnewコマンドで初期化した後に10番、20番の順番でテキストを新規入力していきます。

・行番号 テキスト

先頭に行番号を付加してテキストを入力すると挿入できます。挿入される位置は行番号の順番になります。行番号だけを入力するとその行を削除できます。またすでにある行番号指定するとその行を置換します。行番号にスペース一個だけ入力すると空行を挿入します。1行の文字数は最大で78文字です。また全角文字はサポートしていませんので入力しないで下さい。ESCを二度押すと入力を中止できます。一昔前のBASICと同じ編集方法です。

```
new
10 char a=10;
20 PrNum(a);
list
edit 20
20 PrHex(a);
list
20
list
```

と入力することでテキスト入力コマンドの動作の概要が理解できると思います。

・リナンバー

行間にテキストを挿入し続けるとそれ以上挿入できない状態になる場合があります。この時にはリナンバーをすると行番号が10番刻みになるのでまた挿入が可能になります。リナンバーの方法はsaveで一度ファイルを書き出した直後にloadして下さい。

注意：

編集モードに入るとSilentCは編集用のテンポラルファイルを作成します。その後は1行編集するたびに新たなテンポラルファイルを作成してそれまでのテンポラルファイルを削除するという動作を繰り返します。結果的に編集の最中にはどんどんファイルの無効領域が増加します。編集するファイルが大きければ大きいほどテンポラルファイルも大きくなり無効領域の増加ペースも早くなります。

編集中は一定間隔でdirコマンドを用いて無効領域の大きさを把握してください。また無効領域と使用領域の合計の大きさが800000(800K)を超えた場合には直ちにdefragコマンドで無効領域を最適化して下さい。この際のdefragには数分程度の時間がかかりますのでご注意ください。

1を0にしかプログラムできないフラッシュメモリー上にファイルシステムを構築している故の不便ですがどうかご理解ください。

・実行コマンド

・run

Mainというファイルの中にあるmain()を実行します。引数は渡せません。Main::mainと等価です。ここで注意していただきたいのは実行するのはMainというファイルに書かれているmainという関数だという事です。現在編集中のプログラムではありません。一昔前のBASICでは編集の直後にrunすればすぐにその編集の結果が確認できましたがSilentCはこの点が大きく違います。runは現在編集中のファイルに対するものではなくMainというファイルに対するコマンドです。

SilentCはファイルを読みながら実行するのでrunする前には必ずsaveコマンドで編集結果をファイルに書き戻さなければ変更が反映されません。この違いに戸惑うユーザーの報告も多いので、まずsaveしてからrunするという一連の動作を体で覚えて下さい。

・ファイル名::関数名

指定したファイルの指定された関数を呼び出します。Cの関数としての引数を与える事も可能です。この場合にはファイル名::関数名(引数、引数・・・);という書式になります。この書式はコマンドモードだけではなくプログラム中にも記述できます。要は式によるCの関数呼び出しそのものです。引数を与える必要がない場合には();を省略できます。つまりファイル名::関数名だけでも動作します。

・?コマンド

?の後に式を書くとその結果を10進数で表示します。表示できる範囲は符号付32ビットの範囲です。Cの式であれば何でも記述できます。式の中で関数を呼び出す事も可能です。Cって便利です。

```
?2+2/2
```

と入力してみてください。答えは3になります。2ではありません。

・??コマンド

?コマンドと同様ですが??は結果を16進数で表示します。表示できる範囲は16ビットです。

･ダイレクト実行のサンプル

SilentCで記述可能なすべてのステートメントはダイレクトにタイプすれば即座に実行可能です。しかし一つのステートメントしか実行しませんので複数のステートメントを実行させたい場合には{と}で囲みます。{と}で囲む事で一つのステートメントとして扱われます。ifやforの後に複数のステートメントを実行させたい場合に{と}で囲む事と同じです。つまりCの文法そのものです。
以下はその例です。詳細なSilentCのプログラミング仕様に関してはＯＳ－１プログラミングマニュアルをご覧下さい。

```
PrStr("Hello");
Hello

{int i;for(i=0;i<10;i++)PrNum(i);}
0123456789

RLY1=1;
```

OSIO-1のリレーが動作します。次に上矢印を押してリレーをOFFにしてみて下さい。

```
for(;;)if(!SW1){RLY1^=1;while(!SW1);}
```

SW1を押すたびにリレーが反転します。止めるにはCTRL+Cを押してabortと入力してください。この場合にはリセットを押した方が手っ取り早いです。contと入力すると実行を継続します。

```
{char *a=MemoryAlloc(10);for(;;){Gets(a,10);PrStr(a);PrStr("\r\n");}
```

入力したテキストをそのまま出力します。抜けるにはリセットを押してください。

```
cmd::g
```

cmdというファイルの中にあるg()という関数を実行します。この関数はグローバル変数をダンプするというものです。Mainの先頭に記述されている#include "portdef.h"にはＯＳIＯ－１のスイッチやLEDやリレーなどのポートが定義されているのでそれらのシンボルがグローバル変数として登録されている様子が表示されます。type cmdでソースを覗いて見てください。SilentC内部の変数領域をsizeof関数で求めて表示しています。&で求める変数のアドレスは新たな変数が追加されるとダイナミックに移動しますので&で求めた変数アドレスを利用する際にはすべての変数が生成された後にして下さい。

・中断モードについて

通常のコマンドモードではOKというプロンプトが表示されます。すべてのコマンドが利用可能です。

SilentCの実行中にCTRL+Cを押すと実行を中断して中断モードになります。中断モードでは Break in Mainというプロンプトが表示されます。このモードでは限られた下記のコマンドしか受け付けません。また編集に関するすべてのコマンドが利用できません。abortコマンドで通常のコマンドモードに戻ります。

・dirコマンド
・typeコマンド
・?及び??コマンド
・ダイレクト実行コマンド

・abort

中断モードから編集モードに戻ります。CTRL+Cを押した直後に上矢印キーを押すとこのabortコマンドがプリセットされていますのであとはEnterキーを押せば編集モードに戻ります。実行の中断はCTRL+Cの後に上矢印でEnterと一連の動作で記憶してください。

・cont

中断されていたプログラムの実行を再開します。CTRL+Cで実行を中断した時には?コマンドを利用して変数の内容を調べたり代入文で変数の内容を変更したりしてから実行を再開する事ができます。またプログラム中に#stopという制御文を記述するとCTRL+Cを押さなくても自ら中断モードに入ります。

スタートアップファイル

SilentCインタープリターはユーザーが記述したC言語を一文字ずつ解析しながら実行するインタープリター言語です。SilentCのプログラムはすべてファイルの中に存在しています。通常のインタープリターでは実行するプログラムを一度ファイルからメモリー内にロードしてから実行しますのでこの点が大きく違います。

・スタートアップファイルとは

SilentCで最も重要なファイルはスタートアップファイルです。デフォルトではMainというファイル名が設定されています。このファイル名はレジストリを書き換える事で変更可能ですが基本的には変更しないようにお願いします。またMainというファイルを削除してしまうとSilentCは全く動作しない状態になってしまいますので絶対にMainだけは削除しないで下さい。

C言語の変数はグローバル変数とローカル変数があります。グローバル変数はすべての関数から共通にアクセスできますので関数同士の値の受け渡しに利用します。ローカル変数はその関数の内部でしか利用できません。関数から抜けるとローカル変数はすべて消滅します。

コンパイラ型のC言語であれば関数の外で宣言した変数はすべてがグローバル変数として扱われますが、インタープリター型のSilentCではスタートアップファイル内で宣言した変数をグローバル変数として扱います。これは実行前にすべてのファイルの中のすべての変数宣言をスキャンするにはあまりに時間がかかりすぎるためのSilentCの独自な仕様です。

スタートアップファイル内で宣言した変数はグローバル変数として扱われます。グローバル変数は最小限にしてなるべくローカル変数を利用するようにしてください。

runであってもファイル名::関数名であっても関数の実行前にスタートアップファイルをすべて調べてグローバル変数を生成してから目的の関数を実行します。このグローバル変数のセットアップはダイレクト実行や?などで式を記述した場合などでも行われます。つまりスタートアップファイルで宣言するグローバル変数が多いと実行の開始までに時間がかかってしまったりメモリーを消費してしまいローカル変数が生成できなくなってしまう事になってしまいます。

スタートアップファイル内にmain()という名前の関数を記述すればコマンドモードからrunコマンドで実行できます。もちろんmain()が存在していなくても構いません。runコマンドが動作しないだけです。この場合でもファイル名::関数名(引数);で関数を呼び出す事ができます。

Cコンパイラとの相違点

SilentCは出来る限り標準的なC言語と互換性を保つように製作されましたがメモリー容量の制限やインタープリターで動作するという制約によりいくつかの機能で標準的なC言語と言語仕様が異なっています。高機能なC言語処理系を23KBに圧縮した為の制約と考えて頂ければ幸いです。

・エラー検出が甘い

C言語のプロトタイプ宣言や厳密な型チェックなどはミスを検出するために大変有効な機能ですがSilentCはシンボルが見つからない以外のエラーは基本的に検出しません。また致命的なプログラムミスが隠れていても実行時まではそのエラーは発生しないのでデバッグしにくい言語になってしまっています。

この問題を解決するためにパソコン上にSilentCのシミュレーターを走らせて厳密なエラーチェックを行う「デバッグ文鳥（ぶんちょう)」を併用してPC上でデバッグしてもらう方法を取りました。

・配列の仕様の違い

C言語では配列が利用できます。配列には2種類あって配列宣言をして予め領域を確保して利用する静的な配列変数とポインタ変数に添え字をつけて配列のようにアクセスするポインタ配列です。SilentCでは静的な配列変数をサポートしていません。配列を利用する際にはまずシステムコールで必要なサイズのメモリー領域を確保してそのアドレスをポインタ変数に代入します。その上でポインタ配列として利用して関数を終了する前にその領域を開放するという使い方になります。これはSilentCで利用できる変数領域がわずか2Kバイトしかないのでメモリーを有効に利用するためだとご理解下さい。

・構造体の扱いの違い

SilentCではメモリーのあまり少なさゆえに静的な変数の領域の確保が出来ません。配列と同様に構造体も静的な構造体は使用できません。簡単に言えば'.'(ドット演算子)は使用不能で->(アロー演算子)で構造体にアクセスします。配列と同様にシステムコールで必要な構造体の領域を確保してそのアドレスをポインタ変数に代入します。あとはそのポインタ変数に->を付加して構造体のメンバー名を記述する事で構造体にアクセスするようにして下さい。また'.'演算子は内部的には->として扱われます。構造体の利用に関してはSilentCの独特な仕様になっていますので詳しくはＯＳ－１プログラミングマニュアルのサンプルをご覧下さい。

・変数宣言は実行文である。

SilentCでは変数宣言は実行文としてその都度解釈されます。従ってループの中で変数を宣言するとその都度その変数の削除と生成が行われますので実行時間が遅くなってしまいます。できれば関数の入り口でその関数内で利用するすべてのローカル変数の宣言をしてから実行を開始するようなスタイルでのプログラミングに心がけて下さい。

・#includeや#defineも実行文である。

SilentCでは#include文に出会うとその都度そのファイルをすべて読み込みます。特にループの内部に#includeがあると極端に実行速度が落ちます。また#define文は内部的には文字列変数の宣言文と代入文として処理されています。たくさんのシンボルを#defineするとそれだけローカル変数エリアが増大します。こうした事情から#includeで#defineされたシンボルを読み込む際には変数宣言と全く同様に関数の入り口で#includeして下さい。ここで大切なのは#includeは実行文ですので関数の「中」に記述するという事です。一般的には#includeはプログラムの先頭で読み込んで使う慣わしですが関数の外にいくら#includeや#defineを書いても実行されることは無いので無意味だという事です。

また通常は#includeは使う使わないに関係なくありったけのシンボルを宣言するのが慣わしです。コンパイラであれば参照されなかったシンボルは使われないので#includeファイルはいくら大きくても全く問題にはならないからです。しかしSilentCでは#includeの中に記述した#defineは使う使わないに関係なく、すべて変数として確保されてしまいます。極端な話では#includeだけで変数領域がいっぱいになってしまう可能性もあります。こうした事情から#includeの内部ではその関数内で利用する最低限のシンボルの宣言のみにして下さい。

スタートアップファイル内に#includeや#defineを書くのはなるべく避けて下さい。スタートアップファイルだけは実行に先立って先頭からすべての行を読み込んで実行処理をします。もしスタートアップファイル内に#includeや#defineを書くとそれらはすべてグローバル変数として確保されます。グローバル変数は二度と開放されないので変数領域に居座り続けるからです。また実行開始の際にこうしたファイルを読み込むためにしばらく黙るという事が起き得ます。

また#includeファイル内に実行文を書くと#includeファイルを読み込んだ際に実行されます。これもSilentCの独自の仕様です。

・その他の相違点について

その他にもインタープリター型のＣ言語であるが故の動作の違いが存在しますが、あくまで限られたサイズで実装した言語処理系ですので予め短いプログラムで期待する動作をするかテストしてから実際に利用するようにお願いします。また目立った仕様の違いがあった場合にはお知らせください。マニュアルにて相違点を皆様にお知らせするようにします。基本的にはすべての相違点がSilentCの仕様という事だとご理解ください。

SilentCの詳しい文法やライブラリに関しては別冊のＯＳ－１プログラミングマニュアルをご覧下さい。

・より高度なユーザープログラムの利用

SilentCでは対応できないプログラムを利用したい場合にはSilentCを消去してユーザーが独自に作成したプログラムをフラッシュメモリーに書き込んで利用する事になります。ユーザープログラムエリアとして24kバイトのエリアが確保されています。ユーザーはFreeScaleから提供されているCodeWarriorと呼ばれる統合開発環境を利用してプログラムを開発してＯＳ－１のファイルとして書き込みます。

ファイルを書き込んでからリセットするとSilentMoonはまずユーザープログラムエリアを消去してからファイルを読み込んでフラッシュメモリーにプログラムします。こうした機構を利用してユーザーの作成したプログラムを実行する事が可能です。また工場出荷に戻す操作をする事でいつでもSilentCは復活しますので安心してファームを書き換える事ができます。

しかしサードパーティーから提供されているBDM(Background Debug Module)というプローブを入手するとソースレベルでのステップ動作やトレースやブレークポイントなどの快適な統合環境での開発が可能になりますが、サイレントではこの開発スタイルは基本的にサポートしていません。理由はBDMを利用すると書込み保護されているBootモニター領域も消去が可能になるので結果的に工場出荷に戻す手段を失ってしまうからです。

BDMを利用した開発は大変便利で効率的です。サイレントでもこの方式で開発を行っています。十分にMC9S12NE64について知識を持っていてサイレントから提供されるすべてのソフトウェアを利用せずに自分で最初からソフトを作成する自信のある方は挑戦してみて下さい。そうする事でユーザーがMC9S12NE64をフルに活用できればそれがサイレントの本望です。

ユーザープログラムの開発手順に関しては別冊のＯＳ－１プログラミングマニュアルを参照して下さい。

Bootモニター

Bootモニターはリセット後に最初に動作する2kバイトのプログラムです。シリアル接続を利用してメモリーをダンプしたりフラッシュメモリーを消去・プログラムする機能を提供しています。また工場出荷状態に復帰するリストア機能も提供しています。

・起動方法

Bootモニターの起動方法はまずＯＳＩＯ－１にシリアルターミナルを接続します。シリアルターミナルの設定を57600bps、8ビット、ストップ1ビット、ノーパリティに設定します。フロー制御は「なし」にして下さい。このフロー制御の設定はSilentCの時とは違いますのでご注意ください。Bootモニターは極めてサイズが限られているのでフロー制御を実装できなかったのがその理由です。

次にSW1を押しながらリセットをしてすぐにSW1から手を放します。5秒以上SW1を押し続けていると工場出荷状態への復帰操作になってしまいますので注意してください。

通常はBootモニターは起動する必要はありません。あくまで緊急時にＯＳ－１をなんとか復旧するためのモニターです。

・Bootモニターのコマンド

Bootモニターが起動すると'>'というプロンプトが表示されます。Enterを押すと?が表示されて再度>プロンプトが表示される事を確認して下さい。

・dコマンド

dコマンドはメモリーをダンプします。たとえばd4000とすると4000番地から128バイトのメモリーをダンプします。dとアドレスの間はスペースを空けないで下さい。dのみを入力すると連続的にアドレスを進めながらダンプします。

・eコマンド

eコマンドはBootモニター以外のすべての領域のフラッシュメモリーを消去します。ファイルシステムに利用しているシリアルフラッシュには影響ありません。この操作を行うとSilentMoonとSilentCの双方が消えてしまいますのでご注意ください。万が一この操作を行っても工場出荷に戻す操作を行えばSilentMoonとSilentCは復帰します。確認を求められたらyと答えて下さい。

・gコマンド

gコマンドは指定されたアドレスに実行を移します。gf800とするとf800番地から実行します。gとアドレスの間はスペースを空けないで下さい。

・rコマンド

rコマンドはＯＳ－1を工場出荷状態に戻します。確認を求められたらyと答えて下さい。２０秒程度の時間がかかります。

・Sコマンド

Sコマンドはフラッシュにデータを書き込むコマンドです。Sは大文字ですので注意してください。CodeWarriorで生成されたモトローラSフォーマットをそのまま入力すればチェックサムを確認してSフォーマットで指定されたアドレスにデータをプログラムします。この機能を利用してSilentMoonのアップデートを行います。サイレントからダウンロードした最新のファームウェアはモトローラSフォーマットのHEXファイルですのでそのファイルをターミナルでそのままBootモニターに送る事でフラッシュをプログラムできます。ユーザーはサイレントから提供されるファイル以外のモトローラＳフォーマットファイルを読み込ませないようにお願いします。この場合の動作は全く予測がつきません。