

デジタル・クロックの動作と操作系をプログラミングする

イベント・ドリブンな組み込み系 ソフトウェアの設計とプログラミング

見
本

すでに第5章でプログラムを作成して動かしていますが、ここでは、プログラムの構造と考え方を詳しく解説します。その後、この本の制作課題であるデジタル・クロックの処理を機能ごとに分割し、内容を説明しながらプログラミングしていきます。

処理時間のかかるEEPROMの連続データ書き込み処理は、簡易マルチタスク的な処理で、ほかの処理を妨げないようにしています。

7-1 プログラムの構造

はじめに、C言語プログラムの一般的な構造と、WIZ-Cコンパイラを使った場合の違いなどについて説明します。

main関数

C言語でプログラムを作る場合、必ずmainという名前の関数が必要です。このmain関数はどのようなCコンパイラでも共通で、名前を変えることはできません。リセットが解除されてプログラムが動き出すと、スタートアップ・ルーチンが起動し、特定のレジスタ、初期値付き変数、スタティック変数などへの初期値の設定や、割り込みベクタの設定などの初期化処理が行われた後、main関数が呼び出されます。スタートアップ・ルーチンは、通常はコンパイラが自動的に作成してくれます。プログラマはmain関数の中（または、そこからコールされるサブ関数）に処理を作っていくわけですが、FED WIZ-Cの場合は少し違います。

WIZ-Cアプリケーション・デザイナを使ってプログラムを作る場合、main関数からUserLoopという関数が自動的に呼ばれるようにプログラムのテンプレート(雛型)が作成されます。ユーザはmain関数ではなく、UserLoop関数にプログラムをコーディングしなければなりません。これはmain関数の一部ということになります。

メイン・ループ

組み込みマイコンのプログラムは、ほとんどの場合、電源を切るまで永久に処理を継続するような作り

になっています。処理を輪のように繰り返すのでこれをループ処理と言います。仮にループになっていないとすると、一通り処理を実行した後でmain関数を抜けてしまい、その後は何もできなくなってしまいます。Windowsアプリケーションの場合ならここでアプリケーションが終了するわけです。

このメイン・ループは、処理が何もなくても高速でぐるぐる回っています。図7-1のように、このループの中に何らかのフラグ状態を調べる条件判断を入れ、そのフラグが立っているときだけそれに応じた処理を実行するにすれば、フラグにより処理を切り替えて制御できます。

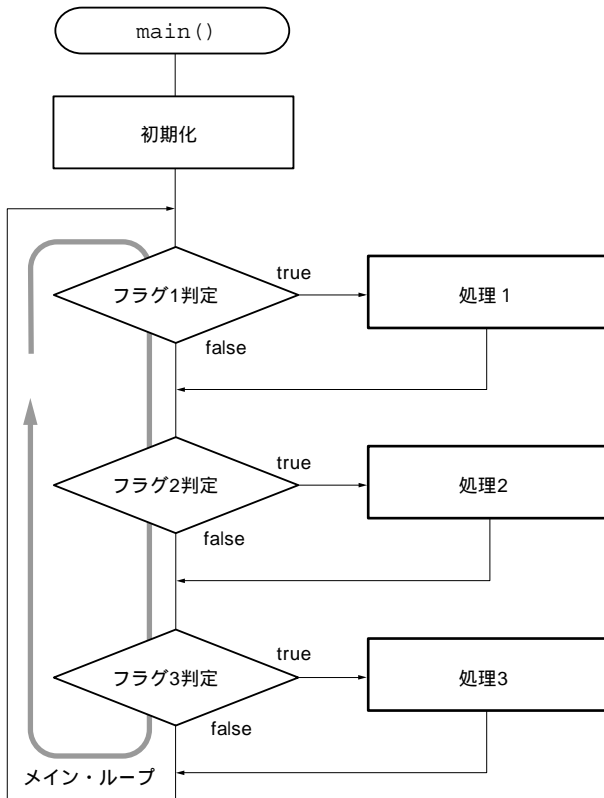


図7-1 処理のフラグ制御
フラグの状態により処理を制御する。フラグは処理1～処理3、または割り込み処理内で何らかの要因で書き換わる（たとえば、1秒経過とかキー入力といった要因）。

Column ... 1 メイン・ループと割り込み処理

割り込み処理はメイン・ループ内の処理とは関係なく起こります(割り込みを禁止していない場合)。割り込みが発生するとメイン・ループ内で実行されている処理(下層のサブルーチンも含む)は一時中断され、割り込み処理に制御が移ります。割り込み処理が

終わると、一時中断されていた処理が再開されます。

割り込み処理がごく短時間で終了すれば、割り込み処理の時間はほとんど無視できます。つまり、割り込み処理とメイン・ループは独立して動いていると考えても支障がありません。

フラグ判定による処理の切り替え

このようにフラグの状態に応じて実行される処理をいくつもループの中に組み込むことで、複数の処理を実行させます。後述する時刻の表示更新やキー入力の判定、タイマ時刻の判定などの処理は、すべてこのような、フラグとその判定処理に基づいて動作します。

フラグが一つも立っていないと、メイン・ループは空回りを続けます。これをアイドル状態と言います。WIZ-C アプリケーション・デザイナの Occurrence 機構はこのフラグの判定処理と、それに応じた処

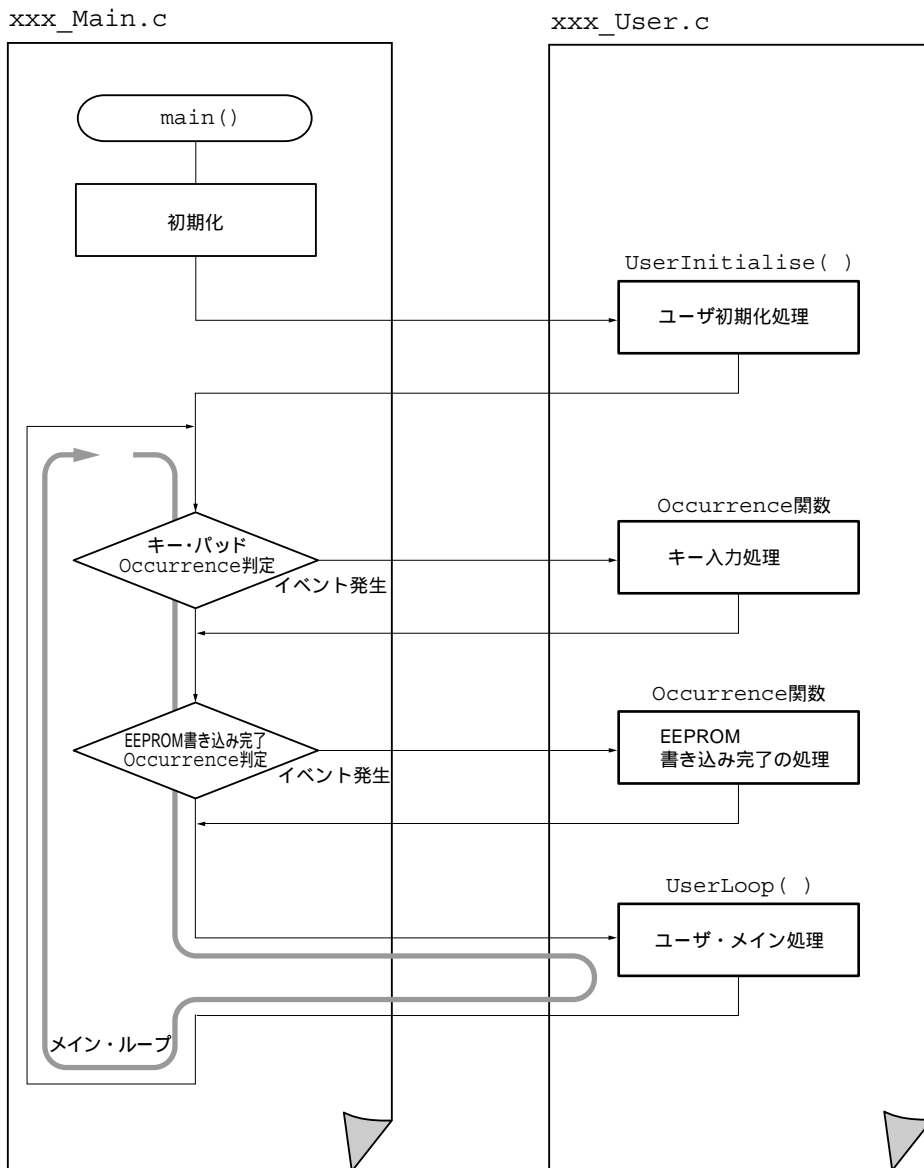


図7-2 Occurrence 判定機構

WIZ-Cのアプリケーション・デザイナを使ってOccurrence機構を利用する場合の一例。Occurrenceの発生を判定するロジックはアプリケーション・デザイナにより自動的に組み込まれる。ユーザはOccurrence関数だけを作成すればよい。

理(Occurrence関数)をWIZ-Cのアプリケーション・デザイナーで半自動的に生成できるようにしたものです。図7-2にはアプリケーション・デザイナーで作成した場合のOccurrence機構の構造を示すフローチャートの例を示します。

タスク

このアイコンは、章末に用語解説があります

このようにフラグを判定して処理を切り替える形態は、大げさに言うとマルチタスクの原理ということができます。この仕組みを汎用的にして、いろいろ機能をもたせたものがリアルタイム・モニタリアルタイムOSと呼ばれるものです。また、このようなフラグのことをイベント・フラグ、処理のことをタスクと呼ぶことがあります。このタスクの実体は関数ですが、この単位で処理をまとめることにより、機能を追加したり変更するのが容易になります。WIZ-CではこのイベントのことをOccurrenceと呼んでいます。

タスクには一つ、大事な約束事があります。それは、タスク内部では長時間の待ち処理を入れてはならないということです。一つのタスクで長時間の待ち状態に陥ると、メイン・ループがそこで途切れてしまうため、ほかのタスクが処理できずに迷惑がかかります。待ち処理というのは、ある意味むだな時間を費やすものです。この間はCPUはひたすら待ちが終わるのを待っているだけです。この待ち時間の間にほかのタスクにCPUの使用権を明け渡せば、その間にもほかの処理ができます。

そこで、待ち処理が発生する場合は、待ちに入るところで自タスクを抜けてほかのタスクへ使用権を譲ります。そして、次のサイクルで再び自分の所にCPUの使用権が回ってきたときに、待ちが解除されていれば、続きから処理を再開します。もし、まだ、待ちが解除されていなければ、また自タスクを抜けてほかのタスクへ使用権を譲ります。

それぞれのタスクをこのような作りにするすることで、メイン・ループは高速に回り、むだなくすべてのタスクが処理をこなすことができます。したがって、タスク内の処理はできるだけ早く終わらせるようにします。処理の必要がなければ、直ちに自タスクを抜けます。今回はEEPROMの連続書き込みにこの手法を使用しています(詳細は後述)。

メイン・ループ内の処理時間

メイン・ループ(UserLoop関数)で実行される処理時間の合計は、時刻更新の1秒より十分に短いということが大前提になります。1秒以内で終わらないと、次の1秒経過のタイミングを取り逃がしてしまいます。また、割り込み処理で仮に1秒以上処理が長引いた場合、当然ですがメイン・ループには絶対に1秒以内には処理が戻ってきません。

このことから、割り込み処理はできるだけ短時間に終わるようにし、メイン・ループ内で長時間かかる処理、待ち処理がある場合も何サイクルかに分けて実行するようにします(前述のタスクの項参照)。

7-2 WIZ-Cプロジェクトの新規作成

それでは、実際に処理を説明しながらプログラムを作成していきます。それぞれの処理の説明に入る前にプロジェクトを新規作成して、エレメントを登録しておきます。第6章で作成した“Key”プロジェクトを元に新規にプロジェクトを作成し、エレメントを追加していきます。新規に作成してもかまいませんが、エレメントやシミュレーション・デバイスなども設定し直す必要があります。プロジェクトの作成手順は

「5-6 キー入力処理のプログラミング(Occurrence関数の使用)」を参照ください。

なお、エレメントの登録順序によりエレメント・ストア・ペインに列挙されるエレメントの順番は変わります。

プロジェクトの新規作成

まず、「5-6 キー入力処理のプログラミング(Occurrence関数の使用)」で作成した“Key”プロジェクトを開きます。メニューの[Project] New Projects use Setting form Current Project をクリックして、現在のプロジェクトの設定を記録します。次に[Project] Open/New Project をクリックして、前回と同様に新しいフォルダを作成し、そこに新しい名前プロジェクトを保存します。

表7-1 エレメントの設定
アプリケーション・デザイナーで設定するエレメントのピン結線とデバイス・プロパティの設定値一覧。












Compare Module1		
	デバイス・プロパティ Operation when registers match Inital Value of Compare Registers	ClearTMR1 15999
Timer1		
	デバイス・プロパティ Enable Timer1 Use internal clock source Synchronise Timer1 input Enable Timer1 oscillator Timer1 prescalar Timer1 read is 16bit	ON ON ON OFF 8 OFF
Timer3		
	デバイス・プロパティ Enable Timer3	OFF
Hex keypad		
	ピン結線 Row1 Row2/Row3/Row4 Col1 Col2 Col3 Col4 デバイス・プロパティ Debounce time in mS Delay befor first repeat (ms) Repeat rate of key (in mS) #of cycles to wait before sampling row Release Row drive after scan Occurrence KP4Pressed	RE0 RE1 RB7 RB6 RB5 RB4 50 1000 100 0 OFF KeyInput
Timer0		
	デバイス・プロパティ Use Prescalar Prescalar Division Clock on Rising Edge Use internal oscillator Enable Timer0 Timer0 is 8bit	ON 4 ON ON ON ON







表7-1 エレメントの設定(つづき)

LCD Module Driver		
	ビン結線 LCDData α (LCDData2 ~ 0は自動結線) LCDE LCDRW LCDRS デバイス・プロパティ Number of lines on LCD display	RD7 RD1 RD2 RD3 2
Data EEPROM		
	デバイス・プロパティ ReadEEPROM at reset Address to read on reset Occurrence EEFin	ON 0 EEDone
Port Driver α (リレー・ポート P ₁)		
	ビン結線 Out1 デバイス・プロパティ Inital value	RC0 0
Port Driver1(リレー・ポート P ₂)		
	ビン結線 Out1 デバイス・プロパティ Inital value	RC1 0
Port Driver α (リレー・ポート P ₃)		
	ビン結線 Out1 デバイス・プロパティ Inital value	RD0 0
Port Driver α (リレー・ポート P ₄)		
	ビン結線 Out1 デバイス・プロパティ Inital value	RE2 0

ここでは、プロジェクト名称を“Clock”とします。プロジェクト・ウィンドウから Key_Main.c と Key_User.c の二つのファイルを削除します(選択してキーボードのDELキーを押す)。

エレメントの追加

すでに“Key”プロジェクトで登録されているエレメントに、さらにエレメントを追加します。アプリケーション・デザイナを表示させて、次のエレメントを追加します。

-  Port Driver1リレー・ポート P₂ 用出力ポート
-  Port Driver2リレー・ポート P₃ 用出力ポート
-  Port Driver3リレー・ポート P₄ 用出力ポート
-  LCD Module Driver ...液晶表示器制御エレメント
-  Data EEPROMEEPROM 制御エレメント
-  Full Digital I/OADCを使わないときに必要なエレメント

既存分も含めすべてのエレメントの結線とプロパティを表7-1に示します。アプリケーション・デザイナでこの表に従って結線し、プロパティを設定してください。