ソフトウェア：DR-Visual Logic

モジュール別のプログラミング：Button, LED

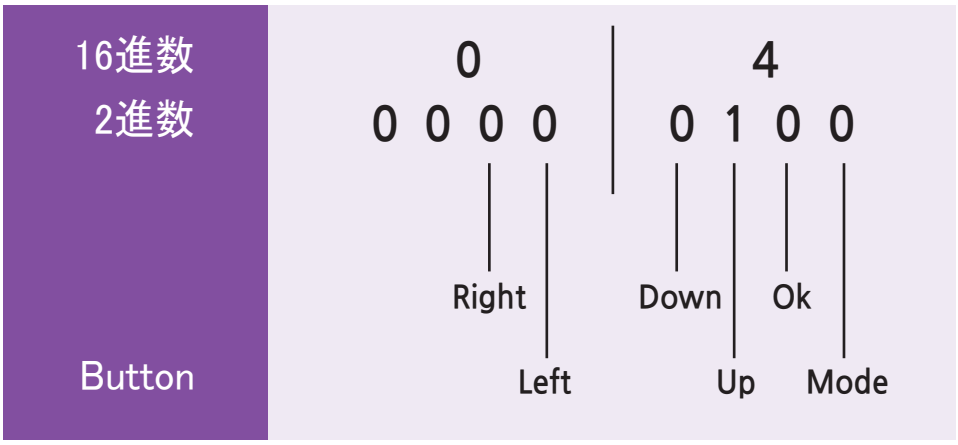
Button、LEDの例

例の説明

DRCコントローラーのButtonを使用して、LEDをオン/オフするプログラミングです。

Button、LEDのプログラミングをするためにまずButtonでLEDを点灯する方法を知っておく必要があります。

$16 = 2^4$



EX) Right

2	0 h
0 0 1 0	0 0 0 0

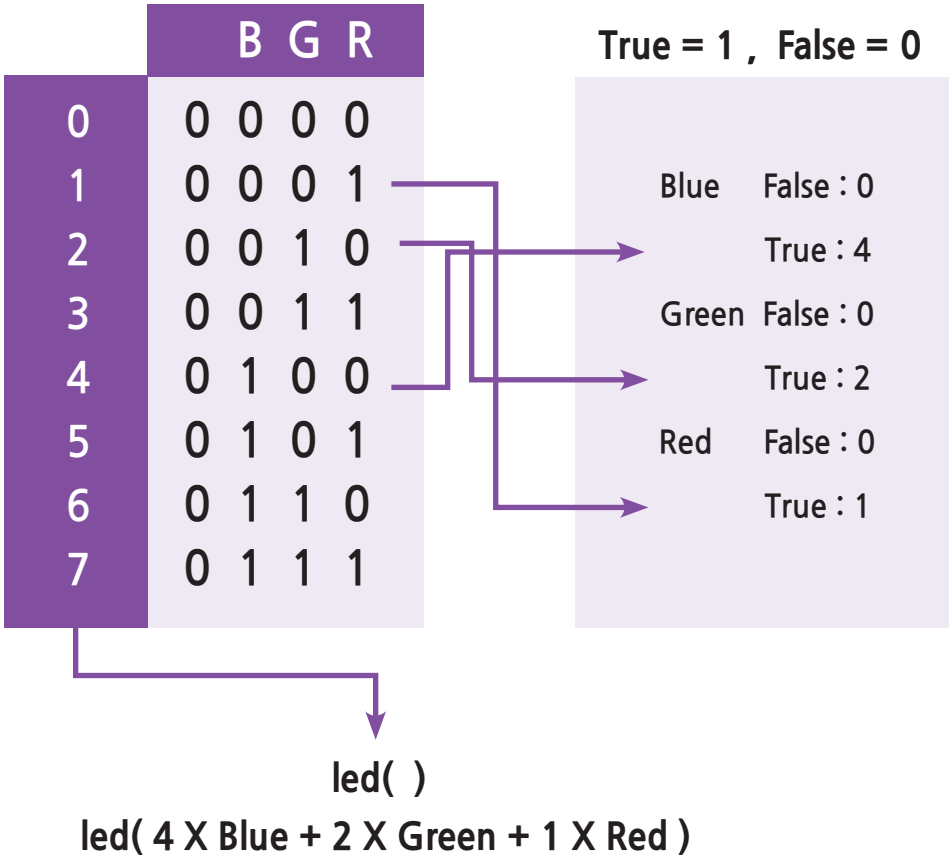
Down

0	8 h
0 0 0 0	1 0 0 0

Button

DRCには6つのボタンがあり、ボタンが押された状態を1Byteで表示します。 1Byteは8Bitで構成されているので、1Byteには8個の1と0を保存することができます。 DRCのボタン6個が押された状態(1)と押されていない状態(0)を表示するためには6Bitが必要です。上図に示すように、各ボタンはBit一つでマッチングしています。

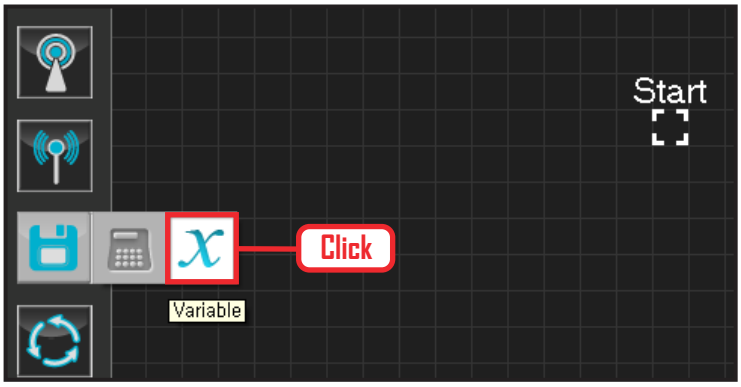
ボタンが押された状態は1と0で表示され、これはButtonのモジュールの右下に16進数で表示されます。 Rightボタンが押された場合、ボタンの値は00100000となり、これを16進数に変更すると20hです(hは16進数という表示です)。 Downボタンが押された場合、ボタンの値は00001000となり、これを16進数に変更すると08hです。もう少し複雑なものとしてUp+Downボタンが押された場合、ボタンの値は00001100となり、これを16進数に変更すると0Chになります。



LED

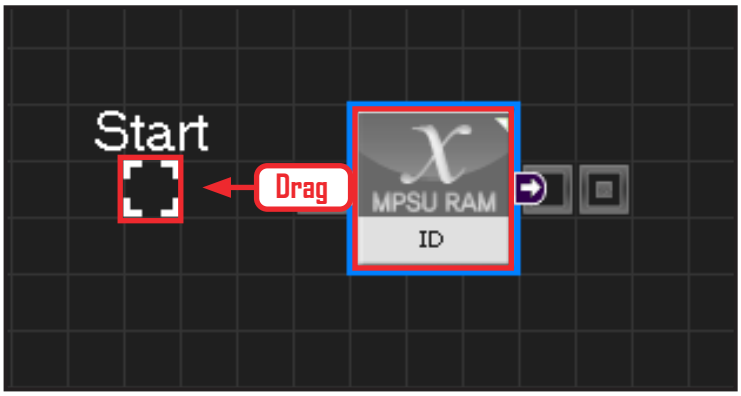
DRCには7つのLEDがありますが、Taskモードで制御可能なのは3つです。 Red、Green、BlueのLED3つのオン状態とオフ状態を表示するために3Bitが必要です。上図に示すように、各LEDはByteの最下位BitからRed、Green、Blueの順にマッチングしています。このLED値をLEDモジュールの入力として入力すると、該当するLEDがオンになります。例えば、LEDモジュールの入力値に2を入力した場合、2は2進数で表示すると、00000010なのでGreen LEDがオンになります。同じ原理で、入力値0(00000000)はすべてのLEDをオフにします。入力値7(00000111)は、すべてのLEDをオンにします。
Blueは2進数では4の位置に相当し、Greenは2の位置、Redは1の位置に対応しています。もし、Blue、Green、Redという名前の変数があり、これらの値(true、false)に基づいて、各LEDをオン/オフするかどうかを決める場合、LEDモジュールの入力値に4x Blue+2 x Green+1 x Redを入れると、格変数名のLEDを制御することができます。
例えば、Blue、GreenがtrueでRedがfalseの時、4 x Blue+2 x Green+1 x Red=6になります。6は2進数で表示すると、00000110ですので、これをLEDモジュールの入力に入力するとGreen、Blue LEDがオンになります。

上記の背景知識に基づいてButtonとLEDのプログラミングをします。



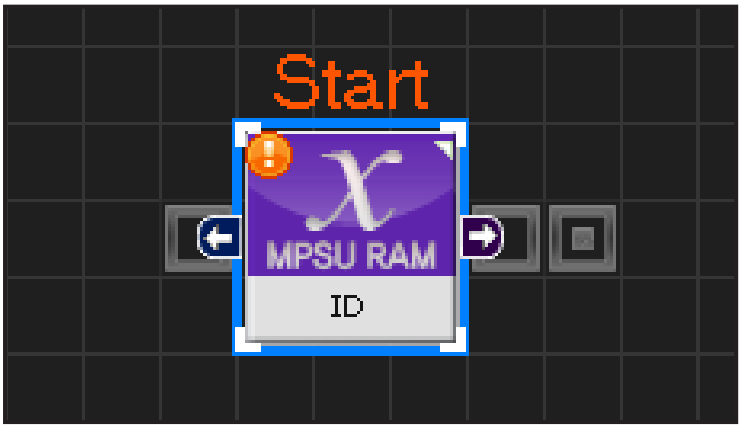
01 変数の指定

Data > Variableモジュールをクリックします。



02 開始

モジュールの左側の連結線をStartPointにドラッグして正確にドッキングさせます。



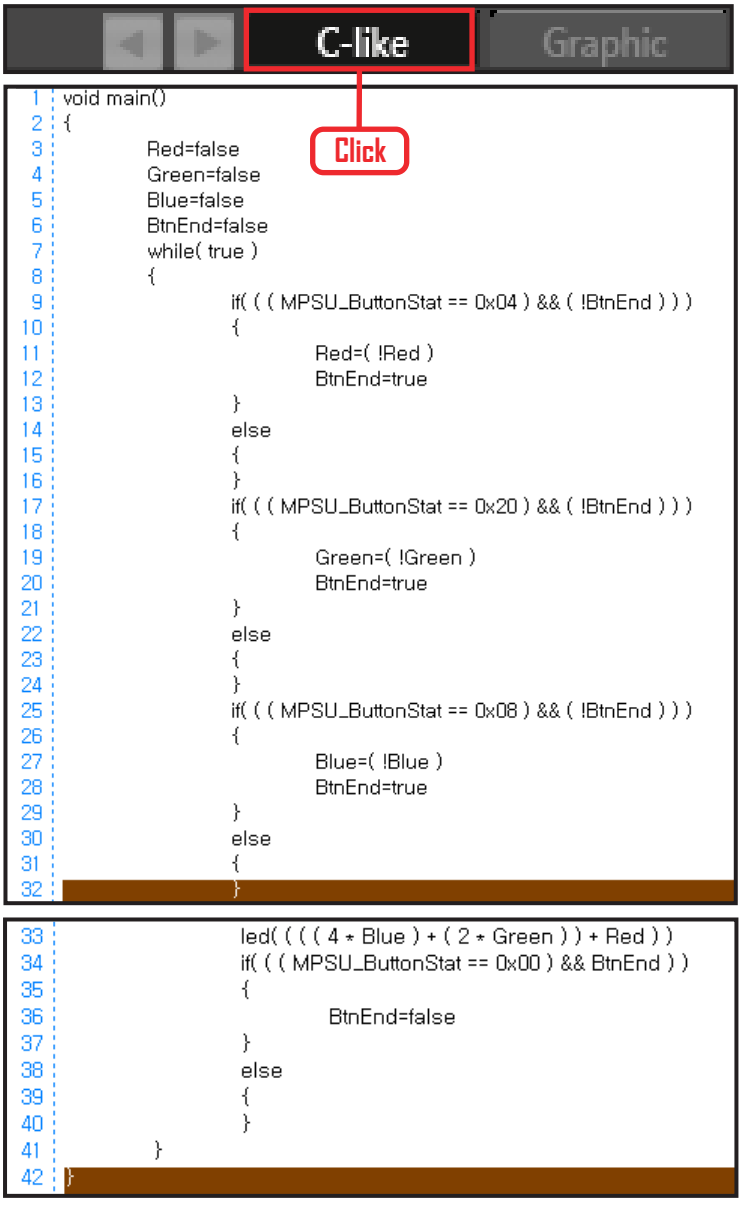
03 プログラミングの開始

モジュールとStart Pointが正確にドッキングすると、左側のように有効化されたカラーイメージのモジュールに変わります。
これはプログラミングが開始されたという意味です。



04 完全プログラミング

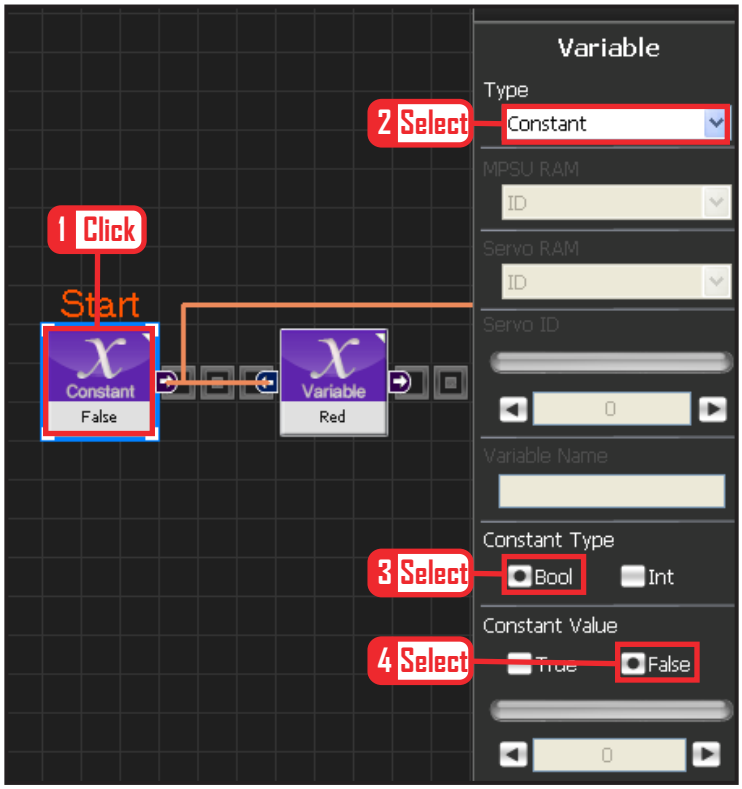
ボタンとLEDを利用した全体のプログラムです。




```
1  void main()
2  {
3          Red=false
4          Green=false
5          Blue=false
6          BtnEnd=false
7          while( true )
8          {
9                  if( ( ( MPSU_ButtonStat == 0x04 ) && ( !BtnEnd ) ) )
10                 {
11                         Red=( !Red )
12                         BtnEnd=true
13                 }
14                 else
15                 {
16                 }
17                 if( ( ( MPSU_ButtonStat == 0x20 ) && ( !BtnEnd ) ) )
18                 {
19                         Green=( !Green )
20                         BtnEnd=true
21                 }
22                 else
23                 {
24                 }
25                 if( ( ( MPSU_ButtonStat == 0x08 ) && ( !BtnEnd ) ) )
26                 {
27                         Blue=( !Blue )
28                         BtnEnd=true
29                 }
30                 else
31                 {
32                 }
33                 led( ( ( ( 4 * Blue ) + ( 2 * Green ) ) + Red ) )
34                 if( ( ( MPSU_ButtonStat == 0x00 ) && BtnEnd ) )
35                 {
36                         BtnEnd=false
37                 }
38                 else
39                 {
40                 }
41         }
42 }
```

05 C-Like

右上のGraphicタブで、C-likeタブをクリックすると、左側のようなTaskプログラミングの画面が表示されます。
ボタンとLEDを利用して全体プログラムをする画面です。
Cと似ている文法構造を持っているのでC文法の先行学習の効果もあります。
各モジュールごとにクリックすると、カーソルが今のプログラムの位置を示しながらモジュールごとにTextにどのように変換するかを確認することができます。

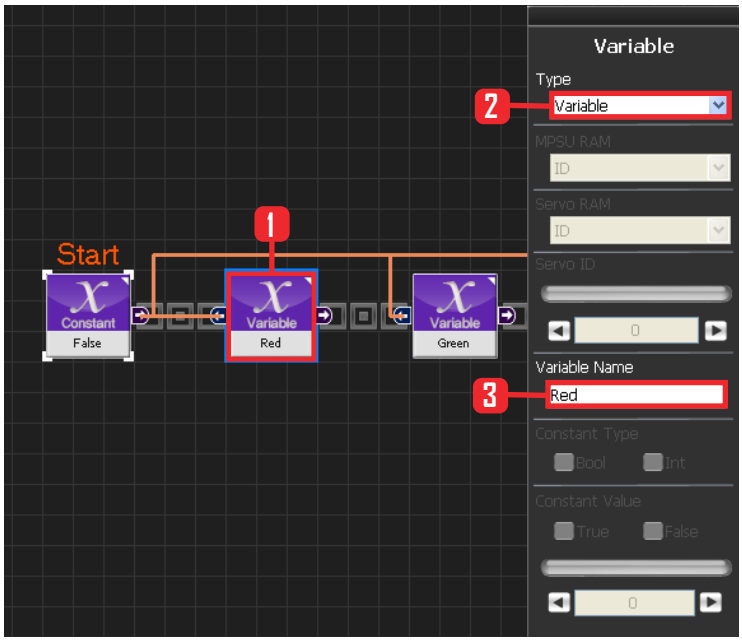


06 Falseに初期化

すべてのLEDをFalse、オフの状態に初期化します。

Data> Variableモジュールを選択します。
Type：Contantを選択します。
Constant Typeは、Boolに設定します。BoolはTrueとFalseを表すデータ型です。
Constant Value：Falseを選択します

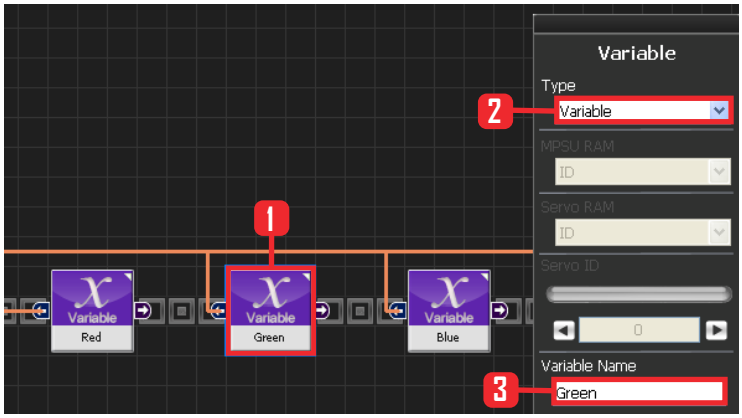
その後、コネクタを使用してFalseをそれぞれの変数に接続します。



07 Red 変数

Data > Variableを選択します。
Type：Variableを選択します。
Variable Name：Redを入力します。

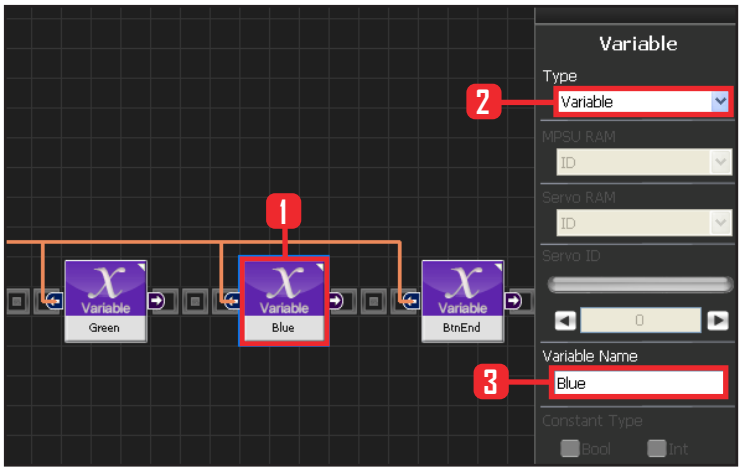
赤LEDはFalseの場合はオン、Trueの場合はオフになります。



08 Green 変数

Data > Variableを選択します。
Type:Variableを選択します。
Variable Name:Greenを入力します。

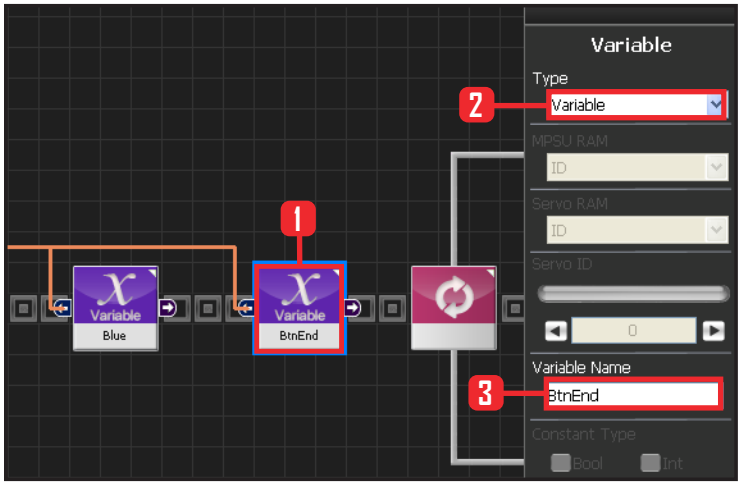
緑LEDはFalseの場合はオン、Trueの場合はオフになります。



09 Blue 変数

Data > Variableを選択します。
Type:Variableを選択します。
Variable Name:Blueを入力します。

青LEDはFalseの場合はオン、Trueの場合はオフになります。



10 BtnEnd 変数

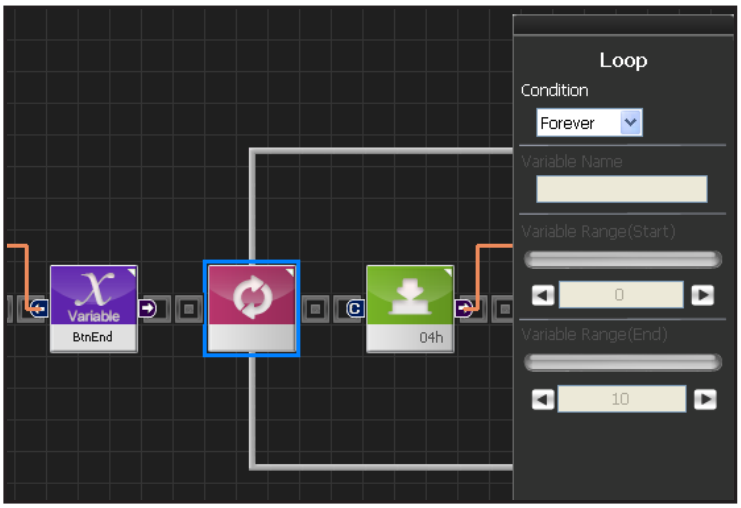
ボタンが押されていない時はFalseのままにしてボタンが押された後、目的の作業が終わる瞬間にFalse -> True に変わる変数です。
また、ボタンを離す瞬間にTrueからFalseに変わる変数です。

Data> Variableを選択します。
Type:Variableを選択します。
Variable Name:BtnEndを入力します。



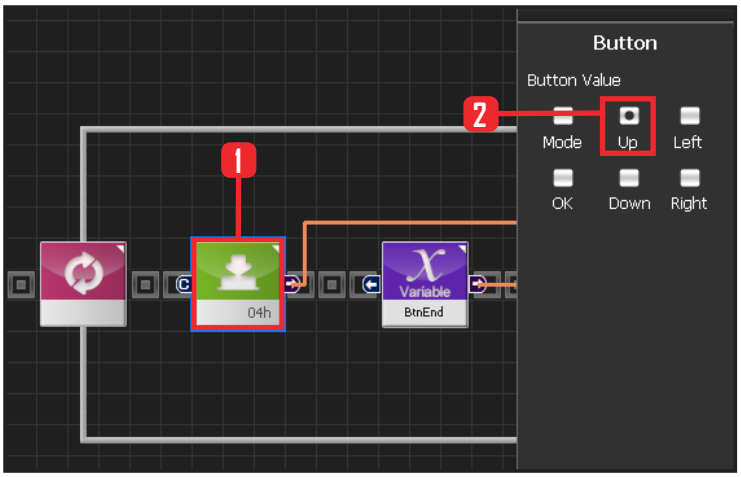
11 変数の宣言

Red、Green、Blue、BtnEnd 4つのすべての初期値をFalseに設定します。



12 Loop

Forever 無限に繰り返します。

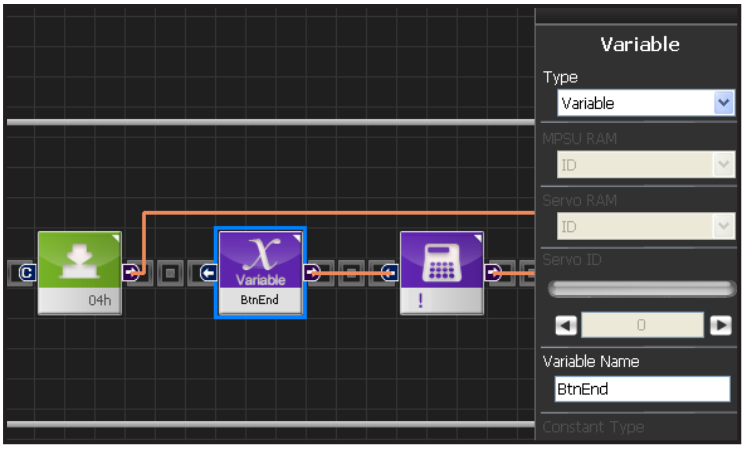


13 Upボタン

ボタンモジュールを作成します。このモジュールは選択したボタンが押された場合はTrueになり、それ以外はFalseになります。Up Buttonを選択すると、Up Buttonが押された場合はtrue、それ以外にfalseになります。

Communication > Buttonモジュールを選択します。
Button ValueのUpを選択します。

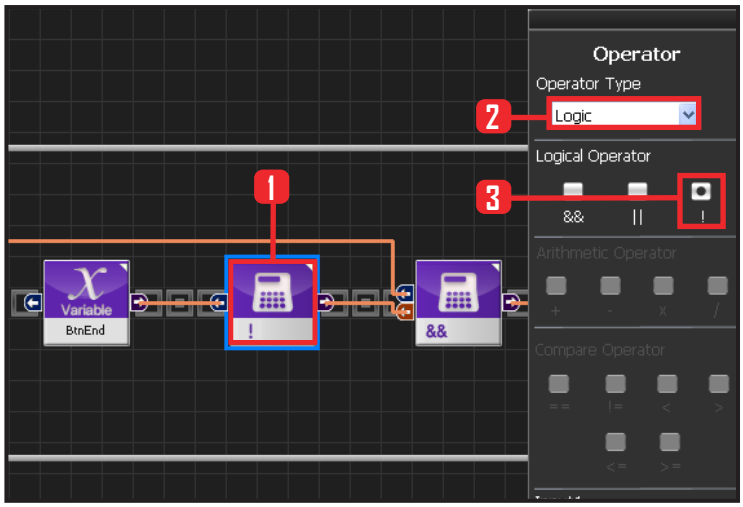
16進数で04hなので、モジュールの画像に04hが表記されます。



14 BtnEnd

BtnEnd値はFalseに初期化されています。後ろにnot演算子がついてTrueになります。

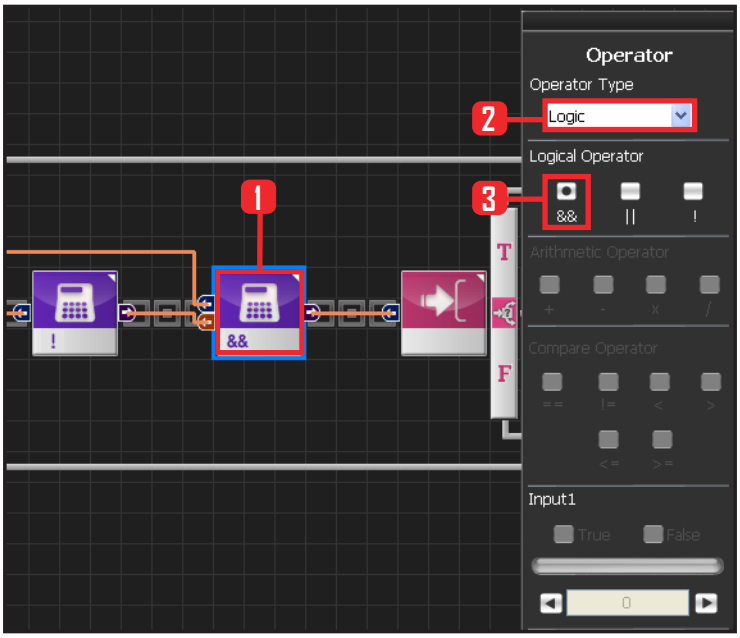
前のbtnEnd変数をコピーして貼り付けます。



15 ！演算

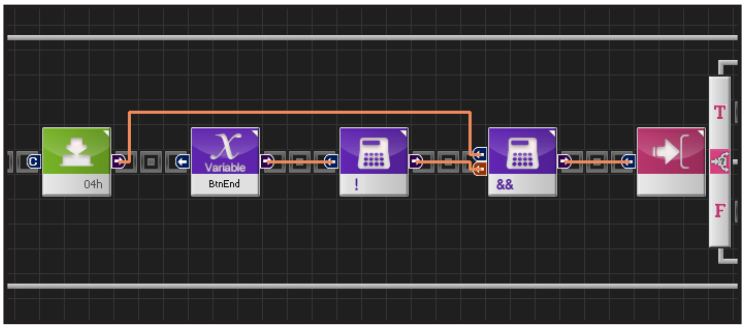
！演算を適用してBtnEndの値を反転します。

Data＞Operatorモジュールを選択します。
Operartor Type：Logicを選択します。
Logical Operator：！を選択します。



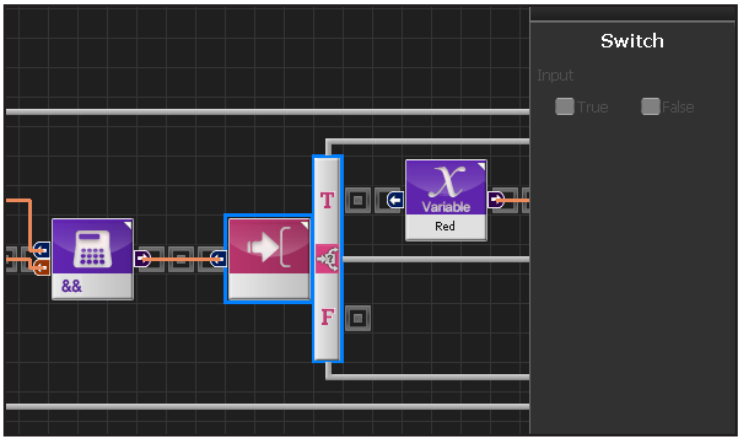
16 And 演算

Upボタンが押され、BtnEndがfalse（！適用して、True）の時、Trueになり、後の条件文を実行します。
Data＞Operatorモジュールを選択します。
Operartor Type：Logicを選択します。
Logical Operator：＆＆を選択します。



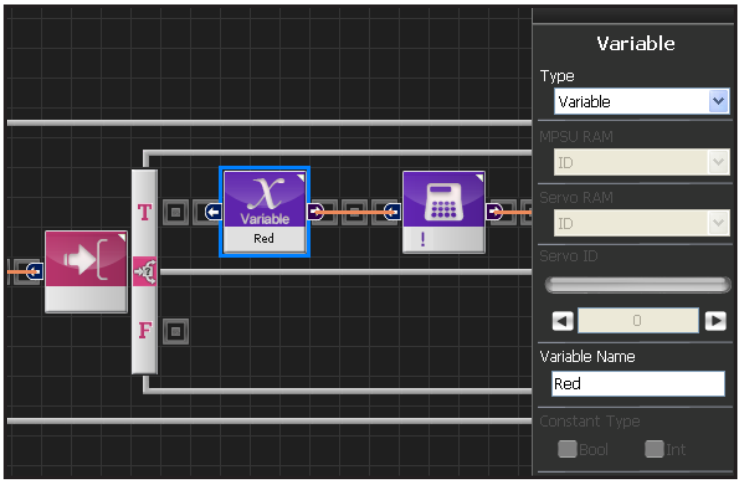
17 Upボタンが押された時

Upボタンが押され、BtnEndがfalseの時、後ろの条件を実行します。



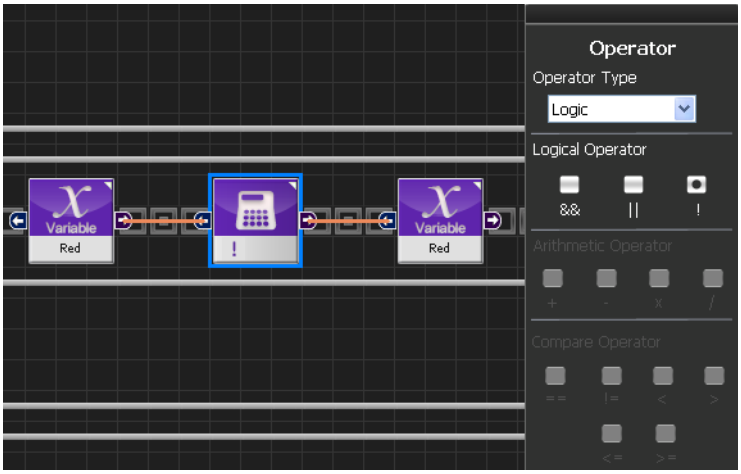
18 If 分岐文

Trueのとき、上の部分を実行します。



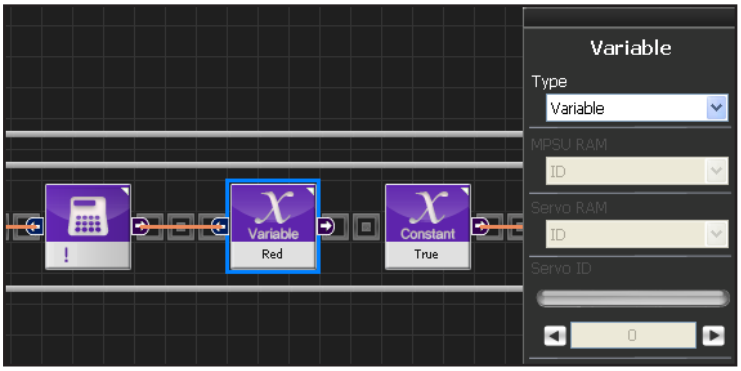
19 Red 出力

前のRed変数をコピーして貼り付けます。



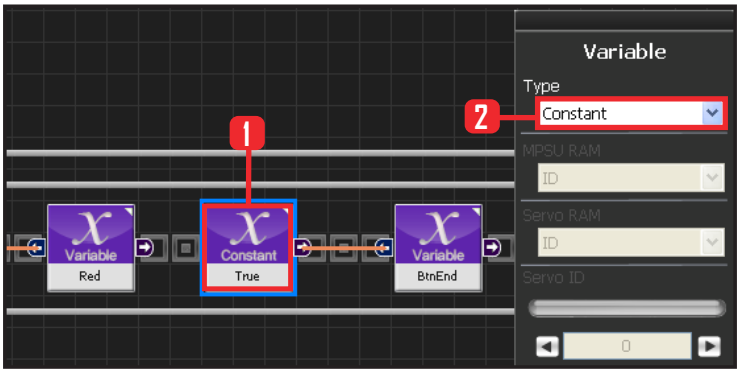
20 ! 演算

RedがTrueの場合はFalseになり、Falseの場合はTrueになります。



21 Red 入力

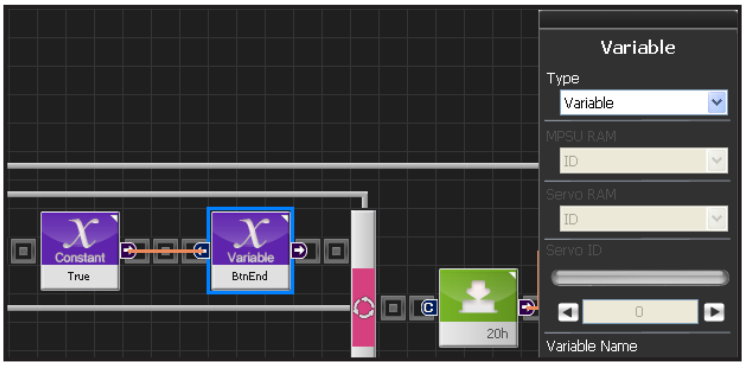
前のRed変数の値がTrueの場合はFalseになり、Falseの場合はTrueに変わって保存されます。



22 True 設定

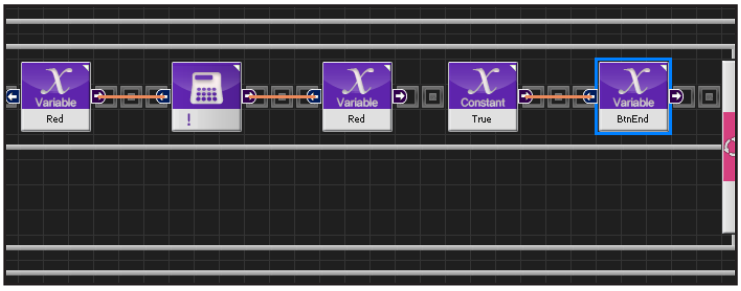
ボタンが押され、関連する動作も終わったのでBtnEndをFalseからTrueに変更しなければなりません。

Data > Variableモジュールを選択します。
Type：Contantを選択します。
Constant Typeは、Boolに設定します。 BoolはTrueとFalseを表すデータ型です。
Constant Value：Trueを選択します



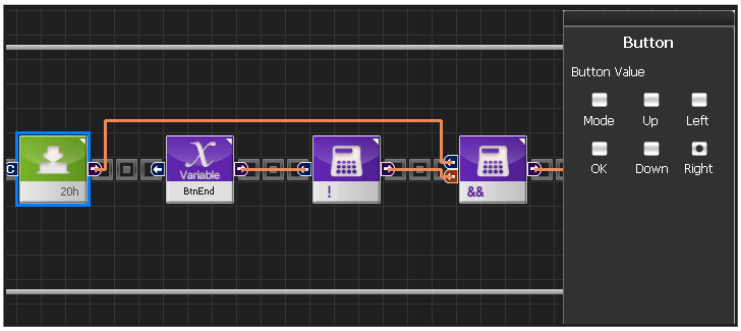
23 BtnEndをTrueに

True値がBtnEndに入力されます。
BtnEndがTrueになると、loop反復時にupボタンが押されていても条件を満たさないのでRed変数の値はこれ以上変わりません。



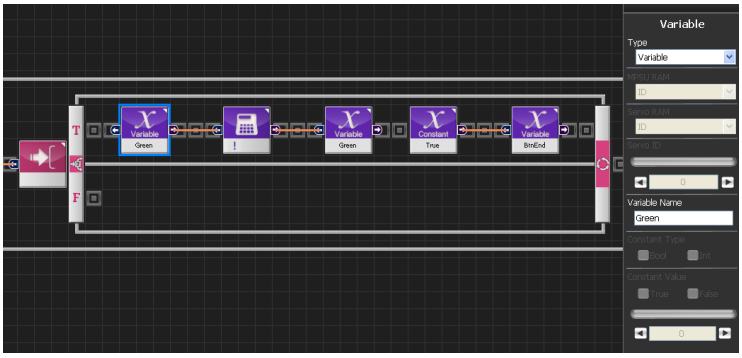
24 Red LED

Upボタンを一度押すと、Redがオンになります。もう一度押すと、Redがオフになります。



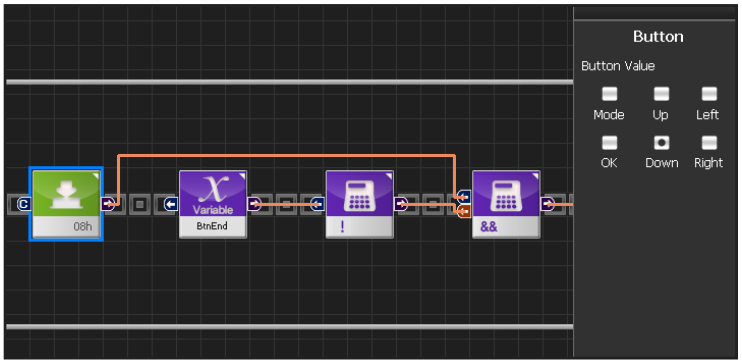
25 Right ボタン

Rightボタンを押した場合です。



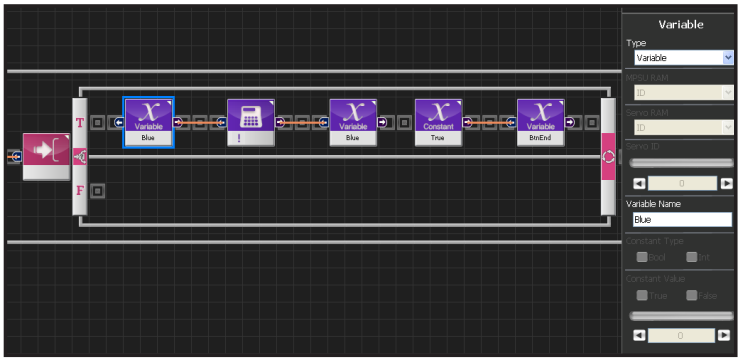
26 Green LED

Rightボタンを一回押すとGreenがオンになり、もう一度押すとGreenがオフになります。



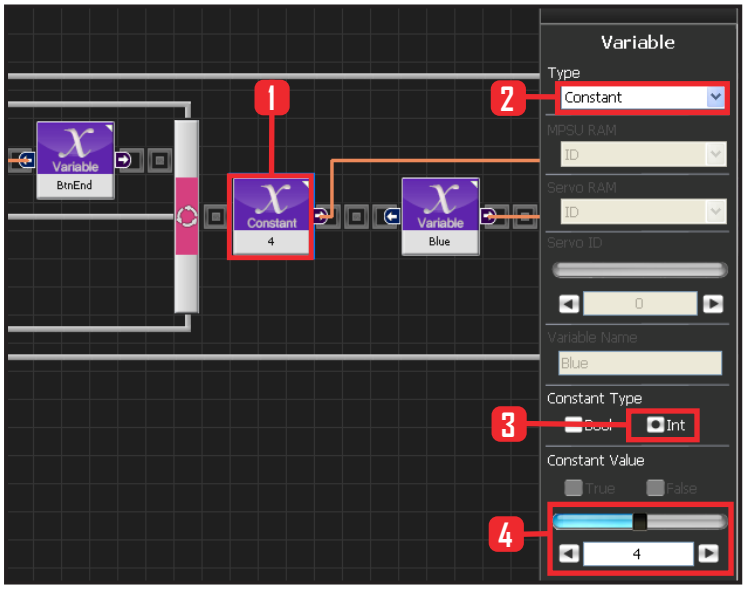
27 Down ボタン

Downボタンを押した場合です。



28 Blue LED

Downボタンを一回押すとBlueがオンになり、もう一度押すとBlueがオフになります。



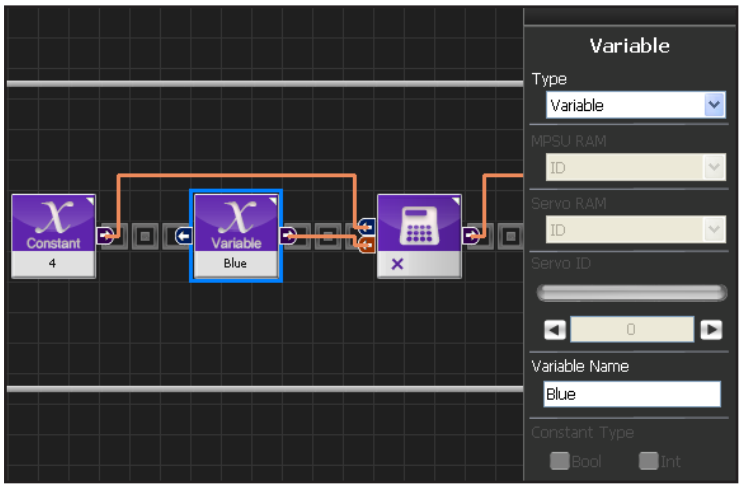
29 LED の値

前述したようにLEDモジュールに値を入力すると、その値に応じてLEDがオンになります。その式は(4 x Blue+2 x Green+1 x Red)です。
上記の式をモジュールごとに接続して表現したのが左図です。

定数4を設定します。

(4 x Blue+2 x Green+1 x Red)

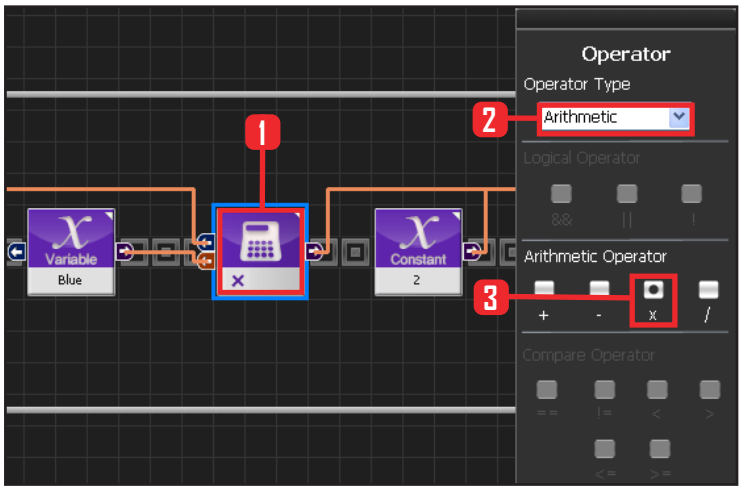
Data > Variableモジュールを選択します。
Type:Contantを選択します。
Constant Typeはintに設定します。
Constant Value:4に設定します。



30 Blue

(4 x Blue + 2 x Green + 1 x Red)

前のBlue変数をコピーします。

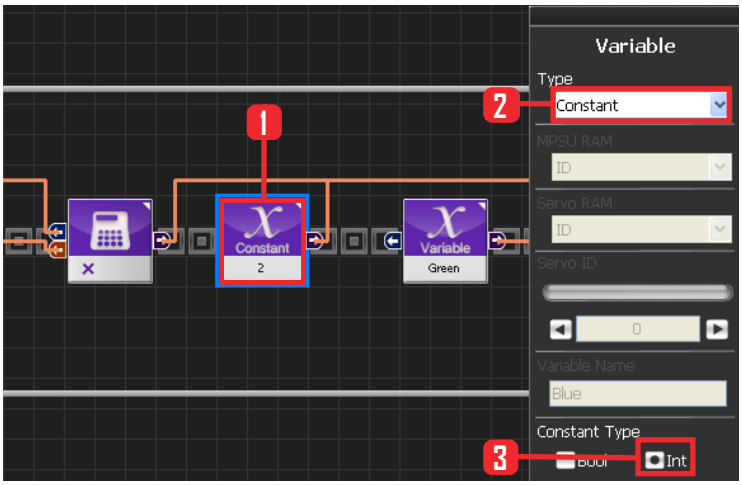


31 掛け算

(4 x Blue+2 x Green+1 x Red)

Data > Operatorモジュールを選択します。
Operartor Type:Arithmeticを選択します。
Arithmetic Operator:Xを選択します。

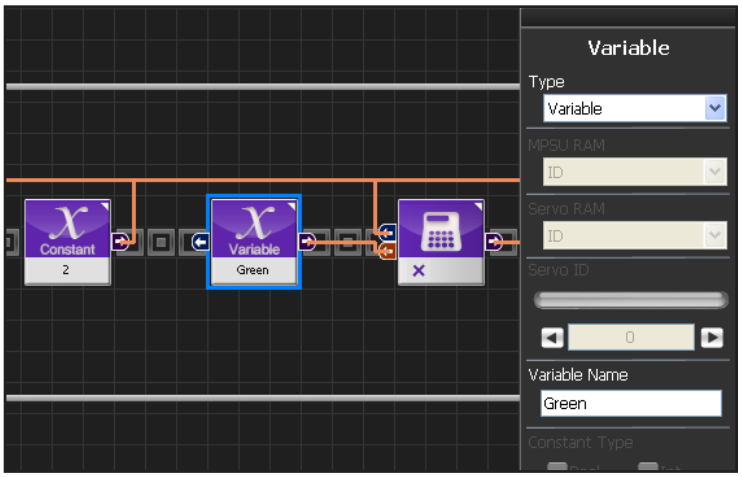
掛け算モジュールの2つの入力コネクタに定数4とBlue変数のモジュールをつなぎます。



32 定数2

(4 x Blue+2 x Green+1 x Red)

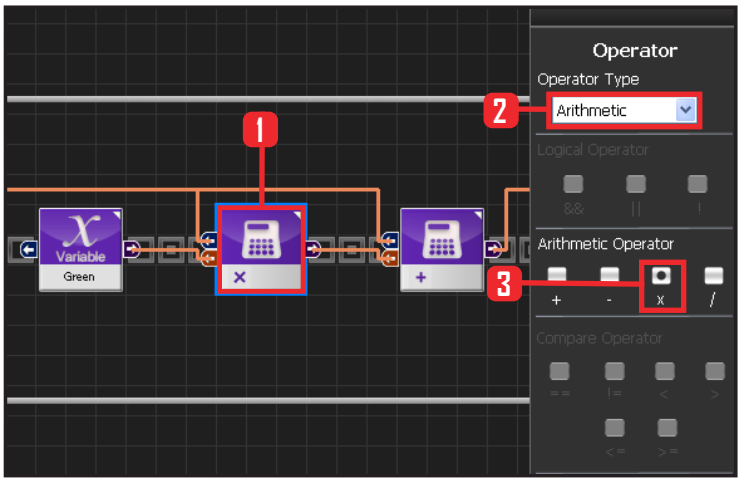
Data > Variableモジュールを選択します。
Type:Contantを選択します。
Constant Typeはintに設定します。
Constant Value:2に設定します。



33 Green

(4 x Blue + 2 x Green + 1 x Red)

前のGreenの変数をコピーします。

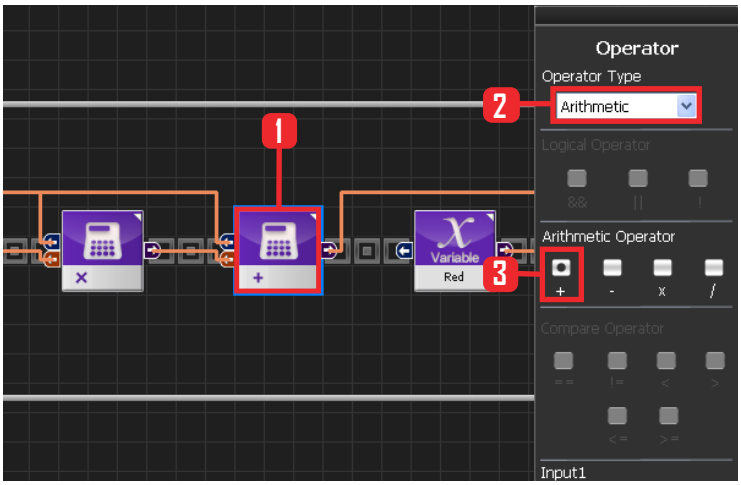


34 掛け算

(4 x Blue+2 x Green+1 x Red)

Data > Operatorモジュールを選択します。
Operartor Type:Arithmeticを選択します。
Arithmetic Operator:Xを選択します。

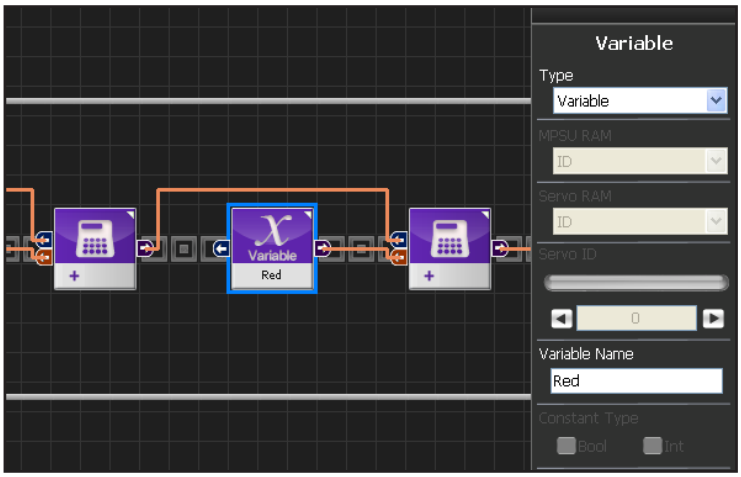
掛け算モジュールの2つの入力コネクタに定数2とGreenの変数のモジュールをつなぎます。



35 足し算

Data > Operatorモジュールを選択します。
Operartor Type：Arithmeticを選択します。
Arithmetic Operator：+を選択します。

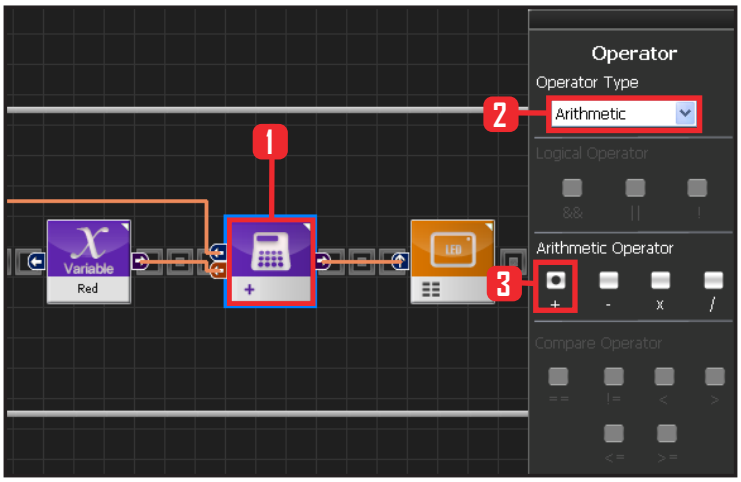
前の31番、34番からの掛け算モジュールの出力を足し算モジュールの2つの入力コネクタに接続します。



36 Red

(4 x Blue + 2 x Green + 1 x Red)

前のRed変数をコピーします。

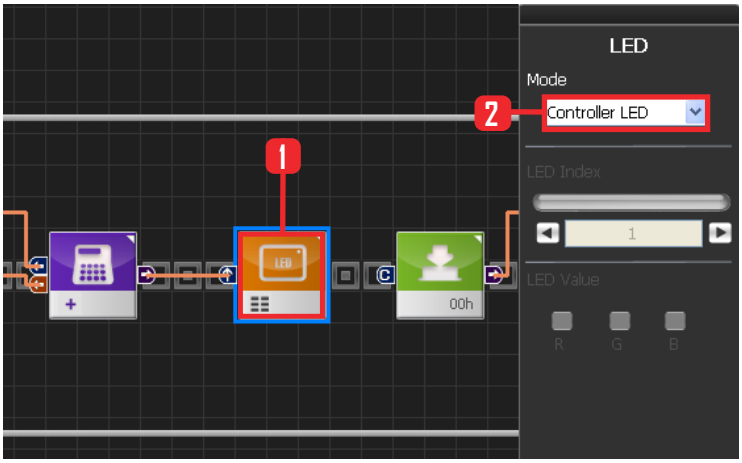


37 足し算

(4 x Blue+2 x Green+1 x Red)

Data > Operatorモジュールを選択します。
Operartor Type：Arithmeticを選択します。
Arithmetic Operator：+を選択します。

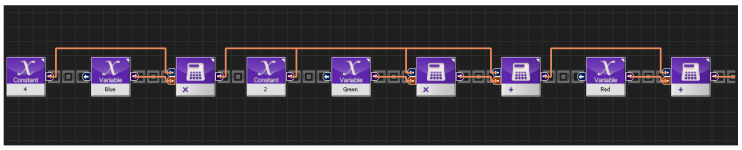
35番の足し算モジュールとRed変数モジュールの出力を足し算モジュールの2つの入力コネクタに接続します。



38 LED

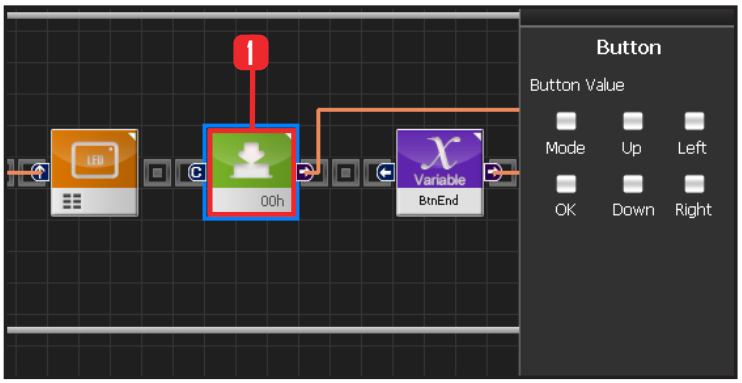
Motion > LEDモジュールを選択します。
Mode：Controller LEDを選択します。

前で演算した値をLED Valueに入れて、それぞれのLEDがオンになります。.



39 LED値を生成する演算

(4 x Blue + 2 x Green + 1 x Red)
をモジュールとして表現したものです。

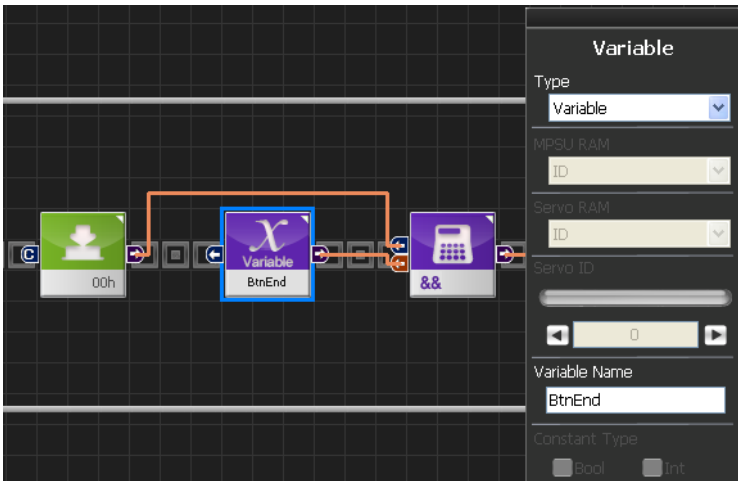


40 ボタンが押されていない状態

ボタンが押され、目的の動作を終えた後、BtnEnd変数はTrueに変更されます。 BtnEnd変数がTrueになると、ボタンに応じた動作をしないため一度押すと必要な動作を一度だけ実行であるようにすることができます。ボタンを離すと、BtnEndをFalseに初期化する必要があります。ここからは初期化過程をプログラミングします。

Motion > Buttonモジュールを選択します。
Button Value：何も選択しません。

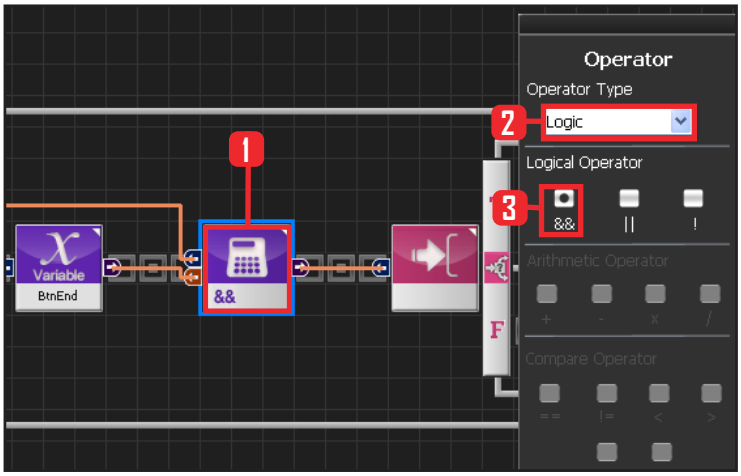
ボタンが押されていない状態を意味します。



41 BtnEndがTrue

BtnEndがTrueの場合です。

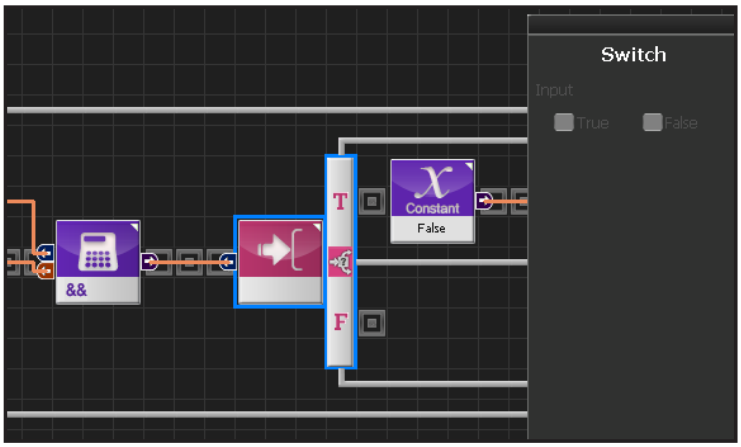
前のBtnEnd変数をコピーします。



42 && 演算子

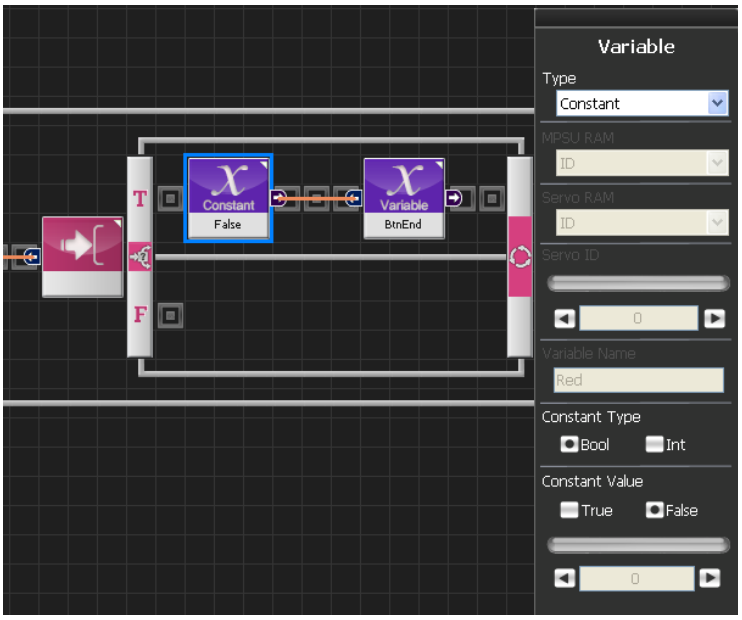
ボタンが押されていない状態とBtnEndがTrueである状態を同時に満足する状態はボタンを離した状態であることを意味します。

Data > Operatorモジュールを選択します。
Operartor Type:Logicを選択します。
Logical Operator: & &を選択します。



43 If 分岐文

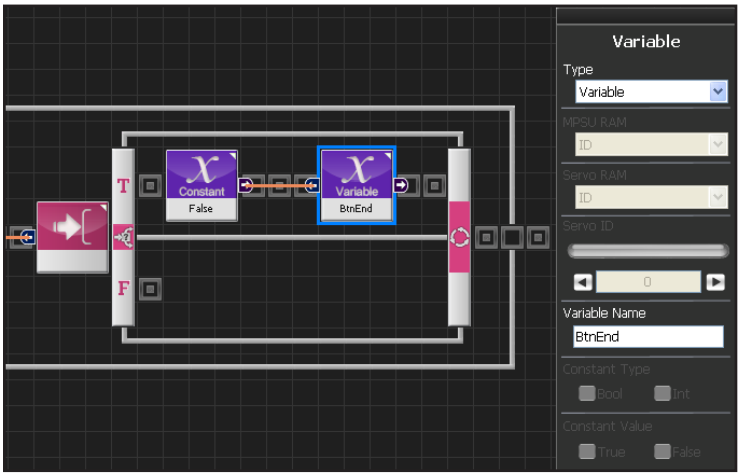
ボタンを押して離すた状態がTrueの場合、実行されます。



44 False 値

BtnEndをTrueからFalseにします。

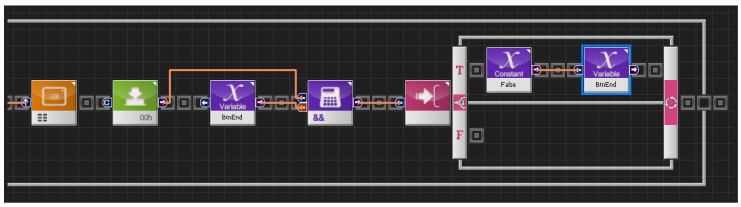
Data > Variableモジュールを選択します。
Type：Contantを選択します。
Constant　TypeはBoolに設定します。BoolはTrueとFalseを表すデータ型です。
Constant Value：Falseを選択します。



45 BtnEndをFalseに

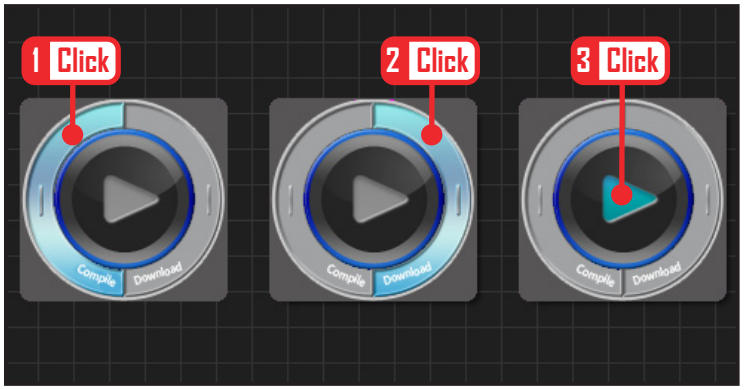
BtnEndにFalse値を入力します。

前のBtnEnd変数をコピーします。



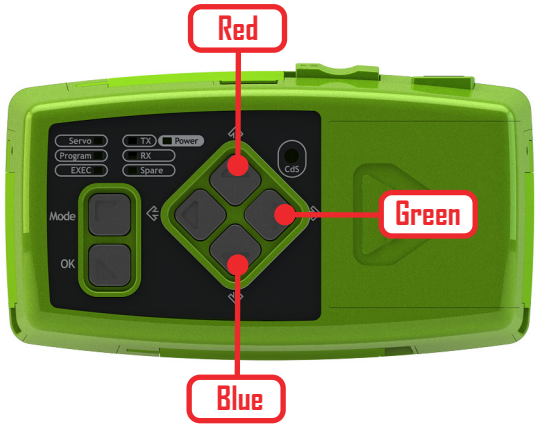
46 ボタンを離した時の初期化

ボタンを離した状態でBtnEndをFalseに初期化します。



47 コンパイル、ダウンロード、実行

左をクリックしてコンパイルします。エラーがなければ右クリックして、ロボットにダウンロードします。ダウンロードが完了すると、中央の矢印の実行ボタンを押して、ロボットで実行させます。



48 ロボット動作

UpボタンはRed
RightボタンはGreen
DownボタンはBlueがオンになります。もう一度押すとオフになります。