~\Downloads\EG_メモ_20131023.c

```
Attack Time:1ms～1024ms
Decay Time :1ms～5120ms
ReleaseTime:1ms～5120ms

ADC->8bit
Interbal->4ms
カーブテーブル->256サンプル

Attackだと....4ms/ADCStep
Decay/Releaseだと....20ms/ADCStep


http://www.keil.com/support/man/docs/c51/c51_library.htm


[構造体]

DBYTE ->8bit
CBYTE ->16bit

/*EG_INFO 稼働するEnvelopeの個数*/
#define EG_INFO_TOTAL_COUNT = 1

/*EG_INFO.bStepClock インターバルタイマー割り込み発生*/
#define EG_STEPCLOCK_WAIT 0
#define EG_STEPCLOCK_INT 1

bit g_iEG_StepClock =   EG_STEPCLOCK_WAIT;  /*インターバルタイマー割り込み発生フラグ*/

typedef struct _st_EG_info {
bit bKey;   /*キーON/OFF*/
bit bLoop;  /*キーON中は繰り返しEnveropeを発生するかフラグ*/
DBYTE iStatus; /*ADSRステータス値*/
CBYTE iAttackValue; /*ADCの値 8bitの値をレフトシフト8bitを行う 0000h,0100h～FF00h*/
CBYTE iDecayValue;
CBYTE iSastenValue;
CBYTE iReleaseValue;
CBYTE iAttackPoint; /*各ADSRステータスのポイント値*/
CBYTE iDecayPoint;
CBYTE iReleasePoint;
CBYTE iAttackStep; /*各ADSRステータスでインターバルタイマー毎にポイント値に加算するステップ値*/
CBYTE iDecayStep;
CBYTE iReleaseStep;
DBYTE iEGLevel; /*EGのDAC値*/
} EG_INFO;

EG_INFO g_stEG[EG_INFO_TOTAL_COUNT];


/*EG_INFO.bKey ポイント値の最大値  0x0FF0=4080 */
#define EG_INFO_MAXPOINT = 4080

/*EG_INFO.iEGLevel EGのDAC値の最大値  0x00FF=255 */
#define EG_INFO_MAX_EG_LEVEL = 255


/*EG_INFO.bKey キーON/OFF*/
#define EG_KEY_OFF 0
#define EG_KEY_ON  1

/*EG_INFO.bLoop キーON中は繰り返しEnveropeを発生するかフラグ*/
#define EG_LOOP_ONESHOTE    0
#define EG_LOOP_LOOP    1

/*EG_INFO.iStatus ADSRステータス値*/
#define EG_ST_ATTACK    0
#define EG_ST_DECAY     1
#define EG_ST_STASTEN   2
#define EG_ST_RELEASE   3

/*EG_INFO.iEGLevel DAC値の最小値,最大値*/
#define EG_LEVEL_MIN    0
#define EG_LEVEL_MAX    255

/*EG_INFO step_EG関数 ADSRステータス変更フラグ*/
#define EG_STEP_STAY        0
#define EG_STEP_CHANGED 1


/*対数テーブル配列:最大インデックス*/
#define CURVE_MAX_INDEX = 255

DBYTE CURVE[]   =   {
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,
  4,  4,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,
  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  9,  9,  9,  9, 10, 10,
 10, 10, 11, 11, 11, 11, 12, 12, 12, 13, 13, 13, 13, 14, 14, 14,
 15, 15, 16, 16, 16, 17, 17, 18, 18, 18, 19, 19, 20, 20, 21, 21,
 22, 22, 23, 23, 24, 24, 25, 25, 26, 27, 27, 28, 28, 29, 30, 30,
 31, 32, 33, 33, 34, 35, 36, 37, 37, 38, 39, 40, 41, 42, 43, 44,
 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 60, 61, 62,
 64, 65, 67, 68, 70, 71, 73, 75, 76, 78, 80, 81, 83, 85, 87, 89,
 91, 93, 95, 97, 99,102,104,106,108,111,113,116,118,121,124,126,
129,132,135,138,141,144,147,151,154,157,161,164,168,172,176,180,
183,188,192,196,200,205,209,214,218,223,228,233,238,244,249,255};


void    initEG  ()  {
    EG_INFO *pEG;

    for ( int i = 0 ; i < EG_INFO_TOTAL_COUNT ; i++ ) {
        pEG =   &g_stEG[i];

        pEG->bStepClock         =   EG_STEPCLOCK_WAIT;  /*インターバルタイマー割り込み発生フラグ*/
        pEG->bKey                       =   EG_KEY_OFF;                 /*キーON/OFF*/
```

```
        pEG->bLoop                  =   EG_LOOP_ONESHOTE;       /*キーON中は繰り返しEnveropeを発生するかフラグ*/
        pEG->iStatus                =   EG_ST_ATTACK;               /*ADSRステータス値*/
        pEG->iAttackValue       =   0x0000; /*ADCの値 8bitの値をレフトシフト4bitを行う 0000h,0010h～0FF0h*/
        pEG->iDecayValue        =   0x0000;
        pEG->iSastenValue       =   0x0000;
        pEG->iReleaseValue  =   0x0000;
        pEG->iAttackPoint   =   0x0000; /*各ADSRステータスのポイント値*/
        pEG->iDecayPoint        =   0x0FF0;
        pEG->iReleasePoint  =   0x0FF0;
        pEG->iAttackStep        =   0x0000; /*各ADSRステータスでインターバルタイマー毎にポイント値に加算するステップ値*/
        pEG->iDecayStep         =   0x0000;
        pEG->iReleaseStep       =   0x0000;
        pEG->iEGLevel               =   0;          /*EGのDAC値*/
    }
}

void    keyOnEG (   EG_INFO *pEG    )   {

    if  (   EG_KEY_OFF  ==  pEG->bKey   )   {
        pEG->bKey                   =   EG_KEY_ON;
        pEG->iStatus            =   EG_ST_ATTACK;
        pEG->iAttackPoint   =   0x0000;
        pEG->iDecayPoint    =   0xFF00;

        pEG->iAttackStep = EG_INFO_MAXPOINT/pEG->iAttackValue;

        pEG->iEGLevel           =   EG_LEVEL_MIN;
    }
}

void    keyOffEG    (   EG_INFO *pEG    )   {

    if  (   EG_KEY_ON   ==  pEG->bKey   )   {
        pEG->bKey   =   EG_KEY_OFF;
        pEG->iStatus = EG_ST_RELEASE;
        pEG->iReleasePoint  =   pEG->iSastenValue;

        pEG->iReleaseStep = EG_INFO_MAXPOINT/pEG->iReleaseValue;

        pEG->iEGLevel   =   pEG->iSastenValue>>4;
    }
}


<intebal> {
    EG_StepClock    =   EG_STEPCLOCK_INT;   /*インターバルタイマー割り込み発生フラグ*/
}


<main roution>
int main {


    for (;;)    {
        if  (   EG_STEPCLOCK_INT == g_iEG_StepClock )   {
            g_iEG_StepClock =   EG_STEPCLOCK_WAIT;
        }
    }
}


[Attack]
bit step_EG_Attack (    EG_INFO *pEG    )   {

    bit iReturn =   EG_STEP_STAY;
    CBYTE   iCurveIndex =   0;

    /*アタックADC値が0ならすぐにDecayに遷移する*/
    if ( 0 == pEG->iAttackValue ) {
        pEG->iEGLevel = EG_INFO_MAX_EG_LEVEL;
        pEG->iStatus = EG_ST_DECAY;
        iReturn =   EG_STEP_CHANGED;

    /*アタックADC値が1以上ならステップ値をポイント値に加算する*/
    } else if ( 0 != stEG.iAttackValue ) {

        pEG->iAttackPoint   =   pEG->iAttackPoint   +   stEG.dAttackStep;

        /*ポイント値が最大値を超えたらDecayに遷移する*/
        if ( EG_INFO_MAXPOINT <= pEG->iAttackPoint ) {
            pEG->iEGLevel = EG_INFO_MAX_EG_LEVEL;
            pEG->iStatus = EG_ST_DECAY;
            iReturn =   EG_STEP_CHANGED;

        /*対数/指数テーブルを指数カーブでDAC値を更新する*/
        }else if ( EG_INFO_MAXPOINT > pEG->iAttackPoint ) {
            iCurveIndex =   CURVE_MAX_INDEX -   ( pEG->iAttackPoint>>4 );
            pEG->iEGLevel = EG_INFO_MAX_EG_LEVEL - CURVE[ iCurveIndex ];
        }
    }

    return  iReturn;
}


[Decay]
if ( 0 == ADCValue ) {
Level = Sasten;
State = Decay;

} else if ( 0 !=ADCValue ) {

double ADCStep = 255/ADCValue;
for ( double i = 255 ; i <= Sasten ; ADCStep-- ) {

if ( Sasten >= i ) {
Level = Sasten;
State = Sasten;
break;
}else if ( Sasten < i ) {
Level = curve[fabs(i)];
}
```

```
}
}


[Sasten]
if ( KEY_ON == Key ) {
if ( 0 == ADCStep ) {
Level = 0;
  Key = KEY_OFF;

} else if ( 0 !=ADCStep ) {
Level = ADCStep;
}
}else if ( KEY_OFF == Key ) {
Level = ADCStep;
State = Release;
}



[Release]
if ( 0 == ADCStep ) {
Level = 0;

} else if ( 0 !=ADCStep ) {
for ( i = Sasten ; i >= Sasten ; abs(255/ADCStep)-- ) {

if ( 0 >= i ) {
Level = 0;
break;
}else if ( 0 < i ) {
Level = curve[i];
}
}
}
```