# Bit Angle Modulation

سلام و درود به تمام دوستان در گروه چیپکده

در این مقاله سعی بر این هست تا با تکنیک جذابی برای توسعه pwm آشنا بشیم.

## صورت مسئله:

احتیاج به تعداد زیادی کانال pwm داریم برای کنترل شدت نور led .مثلاً در حدود ۱۰۲۴ کانال.

عدد بزرگی هست درسته؟ هیچ میکرویی دارای این تعداد pwm یا حتی پین نیست.

کسانی که با شیفت رجیستر 595 کار کردن اطلاع دارن که میشه با این ای سی تعداد پین های میکرو را بسادگی افزایش داد. اما چطور می‌شود از این پین های خروجی pwm گرفت؟ مطلبی هست که در ادامه به اون می پردازیم.

## موارد مورد نیاز:

1) میکرو (هر میکرویی رو می‌شود استفاده کرد. من در اینجا از میکرو معروف و پر کاربرد stm32f0 استفاده میکنم)
2) شیفت رجیستر ۵۹۵ به تعداد مورد نیاز (هر شیفت رجیستر ۸ خروجی دارد با ۲ عدد از آنها می‌شود تعداد ۶۴ led را که به صورت ماتریسی متصل شده‌اند را کنترل کرد)
3) کمی حوصله و البته خواندن این مقاله تا به انتها

## تئوری:

ابتدا لازم هست با مفهوم bit angle modulation که از این به بعد به اختصار bam می‌گوییم. آشنا بشویم.

لازم به ذکر هست که نام دیگر این تکنیک Binary Code Modulation می‌باشد که این نام گذاری خیلی چیزها رو روشن می‌کند.

این بخش رو من از سایت های خارجی ترجمه کرده‌ام پس به کسانی که علاقه‌مند هستند پیشنهادم این هست که یه سرچ ساده در این مورد انجام بدن. کلی مطلب و نمونه کد موجود است که می‌تواند الهام بخش شما باشد. خب بریم سراغ bam :

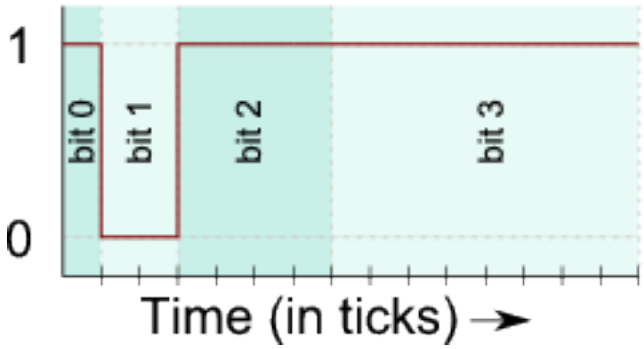
همانطور که می‌دانید زمانی که یک عدد باینری رو از lsb به msb پیمایش کنیم وزن هر دیجیت دو برابر می‌شود اگر ما ۴ بیت داشته باشیم. وزن بیت اول می‌شود ۱ .وزن بیت دوم می‌شود ۲ وزن بیت سوم می‌شود ۴ و در نهایت وزن بیت آخر ۸ میشود.

ما از این خاصیت استفاده میکنیم و به اندازه وزن هر بیت تأخیر ایجاد میکنیم و با روشن و خاموش کردن led با در نظر گرفتن بیت مورد نظر به مقصود مورد نظر می‌رسیم.
با یه مثال همه چی روشن تر می‌شود:

1101

عدد باینری بالا رو درنظر بگیرید . که برابر است با عدد ۱۳ در مبنای ده دهی.

اگر بخواهیم led را با دیوتی سایکل ۱۳/۱۵ روشن کنیم با توجه به شکل زیر :




1. led را به اندازه ۱ تیک روشن می‌کنیم. وزن بیت اول ۱ هست.
2. Led را به اندازه ۲ تیک خاموش می‌کنیم. وزن بیت دوم ۲ هست.
3. Led را به اندازه ۴ تیک روشن می‌کنیم. وزن بیت سوم ۴ هست.
4. Led را به اندازه ۸ تیک روشن میکنیم. وزن بیت چهارم ۸ هست.

و همین پروسه رو تکرار میکنیم.

پس در هر دوره زمانی ۱۵ تیک (۴ بیتی) led مورد نظر ما به اندازه ۱۳ تیک روشن می‌ماند که برابر با دیوتی سایکل ۸۶ ٪ می‌باشد.

(13/15) * 100 = 86%

این پروسه ۴ بیتی به راحتی قابل افزایش به ۸ بیت یا بیشتر است.

مقدار tick به فرکانس bam و رزولوشن آن مربوط می‌شود و با فرمول زیر قابل محاسبه است:

tick * bit resoloution = 1/bam frequency

اگر مقدار تیک رو ۱ میلی ثانیه در نظر بگیریم فرکانس پایه bam با ۴ بیت رزولوشن می‌شود:

1ms * 16 =16ms = 62.5hz

که برای کنترل led مناسب است.

بچه‌های باهوش متوجه شدن که یکی از محدودیت‌های این روش رزولوشن bam هستش و هر چه رزولوشن بالاتر برود احتیاج به cpu قوی‌تر و کلاک بالاتر است. اما تعداد led ها تأثیر زیادی بر نحوه عمل‌کرد الگوریتم ندارد.

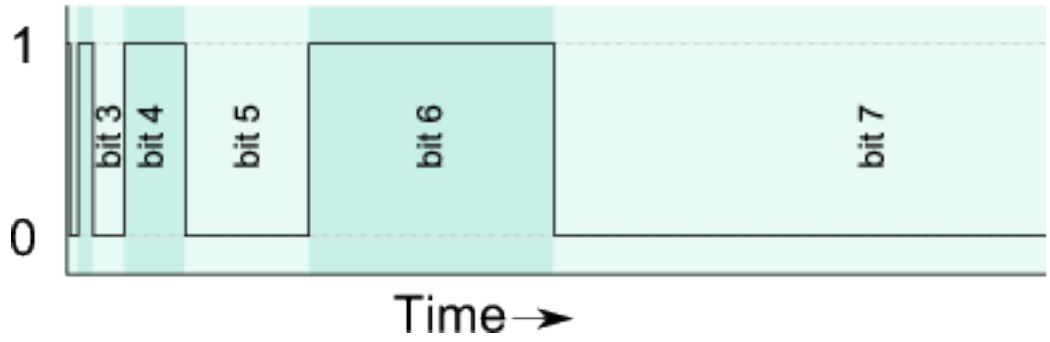
همین خصوصیات باعث می‌شود این تکنیک برای کنترل تعداد زیاد led بطور چشمگیری  عالی یاشه اما در عمل برای کنترل سرو موتور که احتیاج به رزولوشن بالاتری داره کارایی مطلوب رو نداشته باشه.

یک مثال ۸ بیتی رو هم ببینیم و بعدش بریم سراغ شماتیک و کدنویسی.

برای روشن کردن led با دیوتی سایکل ۳۳٪  با در نظر گرفتن رزولوشن ۸ بیتی :

33% = 85 / 255

عدد مقدس ۸۵ در مبنای دو میشود: ۰۱۰۱۰۱۰۱ به شکل زیر توجه کنید:




1. led رو به اندازه ۱ تیک روشن میکنیم.
2. led رو به اندازه ۲ تیک خاموش میکنیم.
3. led رو به اندازه ۴ تیک روشن میکنیم.
4. led رو به اندازه ۸ تیک خاموش میکنیم.
5. led رو به اندازه ۱۶ تیک روشن میکنیم.
6. led رو به اندازه ۳۲ تیک خاموش میکنیم.
7. led رو به اندازه ۶۴ تیک روشن میکنیم.
8. led رو به اندازه ۱۲۸ تیک خاموش میکنیم.

و مراحل بالا رو تکرار میکنیم.

به این ترتیب led ما به مدت زمان ۸۵ تیک  روشن و ۱۷۰ تیک خاموش بوده. با توجه به پریود کامل ۲۵۵ تیک . برابر با ۳۳٪ پریود که مطلوب ما بوده است.

با توجه به مراحلی که در بالا اشاره شد پیاده‌سازی الگوریتم رو می‌شود به شکل زیر بیان کرد:

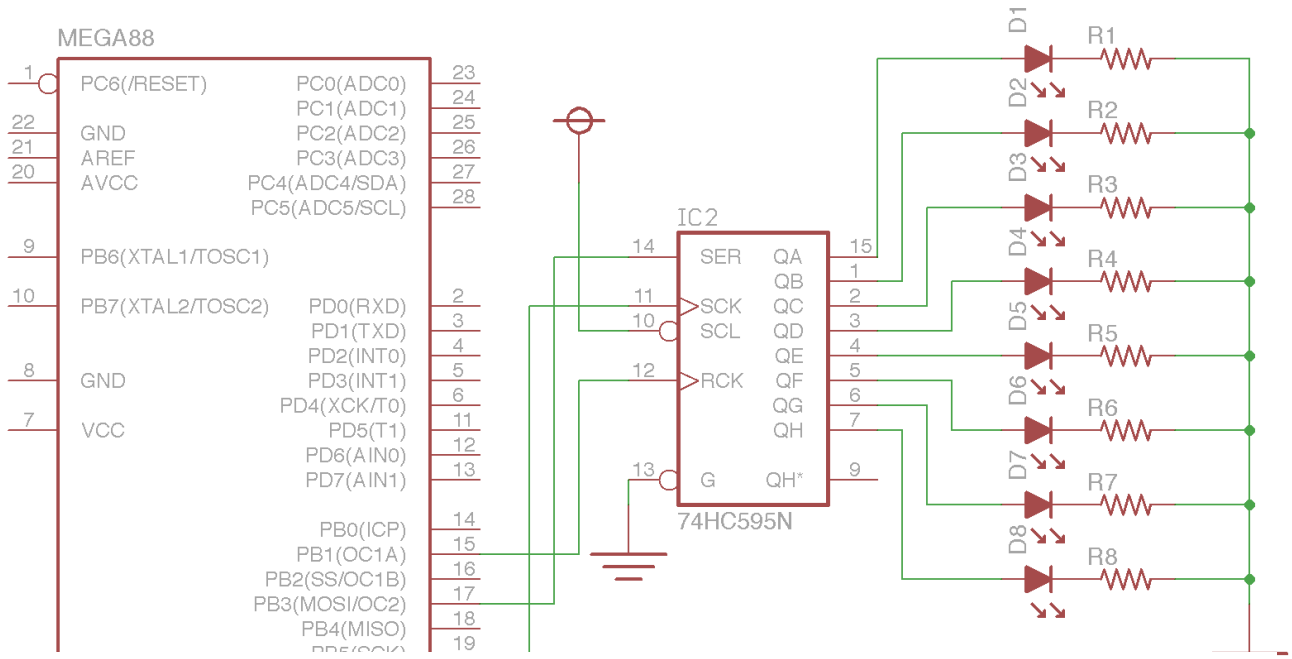
1. led رو برحسب بیت n ام ست میکنیم.
2. اندازه ۲ به توان n تأخیر ایجاد میکنیم.

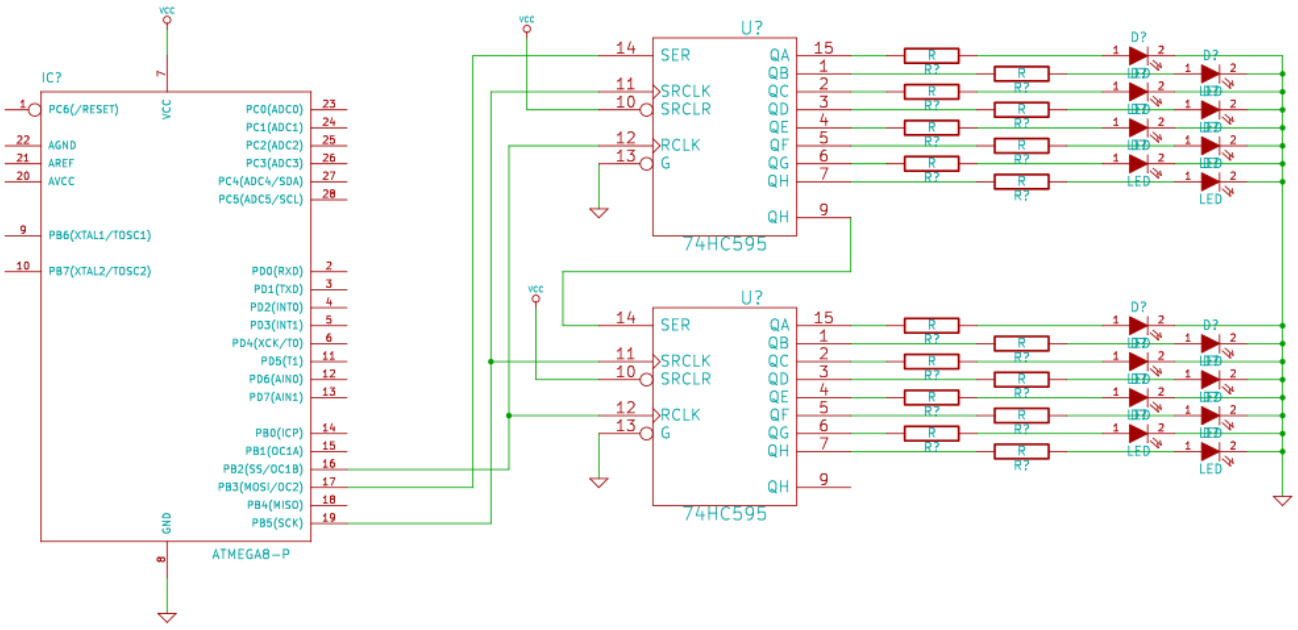
# شماتیک:

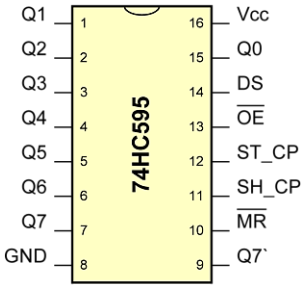
برای متصل کردن شیفت رجیستر ها به میکرو من از واحد spi استفاده میکنم.پایه SER به mosi و پایه SCK به کلاک spi متصل می‌شود و یک عدد gpio لازم داریم برای متصل کردن پین latch (RCLK)

پایه ۹ خروجی سریال می‌باشد که میتوان به پایه SER شیفت رجیستر بعدی متصل گردد برای cascade یا سری کردن شیفت رجیستر ها.

به شماتیک های زیر توجه کنید:






**74HC595 8-Bit Shift Register Pinouts**

| Pin | Symbol | Description |
|---|---|---|
| 1 | Q1 | Parallel data output (bit-1) |
| 2 | Q2 | Parallel data output (bit-2) |
| 3 | Q3 | Parallel data output (bit-3) |
| 4 | Q4 | Parallel data output (bit-4) |
| 5 | Q5 | Parallel data output (bit-5) |
| 6 | Q6 | Parallel data output (bit-6) |
| 7 | Q7 | Parallel data output (bit-7) |
| 8 | GND | Ground (0 V) |
| 9 | Q7` | Serial Data Output |
| 10 | $\overline{MR}$ | Master Reset (Active Low) |
| 11 | SH_CP | Shift Register Clock Input |
| 12 | ST_CP | Storage Register Clock Input |
| 13 | $\overline{OE}$ | Output Enable (Active Low) |
| 14 | DS | Serial Data Input |
| 15 | Q0 | Parallel data output (bit-8) |
| 16 | Vcc | Positive Supply Voltage |

## کد نویسی:

به مرحله شیرین کد نویسی رسیدیم :))

1. احتیاج به یک تایمر داریم که مقدار تیک رو محاسبه کنه و بتونیم دیلی ایجاد کنیم.
2. یک واحد spi لازم داریم برای انتقال دیتا به زنجیرهٔ شیفت رجیستر ها
3. یک gpio لازم داریم برای پایه لچ شیفت رجبستر
4. استیت ماشین برای پیاده‌سازی الگوریتم bam
5. واحتمالاً چند انیمیشن برای اینکه کد رو تست کنیم

در ابتدا استیت ماشین رو طراحی میکنیم در محیط کنسولی و بعد از جواب گرفتن. کد رو به eclipse منتقل میکنم. و باقی کار ها رو آنجا ادامه می‌دهم.

من رو ماشین لینوکسی در حال کد نویسی هستم و یه همین خاطر از eclipse استفاده میکنم. اگر شما از ویندزد :)) و iar استفاده میکنید کافیه یه پروژه بوسیله cube بسازید و فایل ها منتقل کنید ودوباره کامپایل کنید که مطمئن هستم تو اینکار شما یک نابغه هستید :))

## استیت ماشین:

ابتدا چند دیفاین انجام میدم تا زندگی برای همه آسون تر باشه.

```
typedef unsigned char uint8_t;

#define number_of_shift_register    2
#define number_of_shift_register_outputs    8
#define number_of_pwm_channel (number_of_shift_register * number_of_shift_register_outputs)

#define BAM_RESOLUTION      4
#define BAM_NUMBER_OF_STATE 4

#define BAM_STATE_0_MAX_COUNTER 1
#define BAM_STATE_1_MAX_COUNTER 2
#define BAM_STATE_2_MAX_COUNTER 4
#define BAM_STATE_3_MAX_COUNTER 8
```

در مورد ۴ دیفاین آخر لازم به توضیح هست که مقدار دیلی یا وزن هر بیت رو تعریف کردم. باقی کد ینظرم کاملاً گویا هستش و احتیاج به توضیح نداره.

حالا یک enum داریم که استیت های ما رو مشخص میکنه. چون رزولوشن رو ۴ بیت در نظر گرفتیم پس ماشین ما دارای ۴ استیت می باشد.

```
typedef enum {
    BAM_STATE_BIT_0 = 0,
    BAM_STATE_BIT_1,
    BAM_STATE_BIT_2,
    BAM_STATE_BIT_3,
} BAM_STATES_T;
```

یک wrapper ایجادمیکنیم برای تابع ارسال دیتا به زنجیره شیفت رجیستر ها تا تابع send_data کپسوله بشه واگربعدا احتیاج بود. از روشی غیر از SPI هم بتوان دیتا را ارسال کرد. البته این تابع رو بصورت پوینتر تعریف میکنیم تا به طور هیجان انگیز تری ازش استفاده کنیم.

```
/*  wrap the function which is functionallity is sending data to the shift register chain
 *  so we can use both SPI or GPIO function call*/
typedef void (*bam_state_send_data_fp) (BAM_Handler_Ptr);
```

حالا یه استراکت و یه پوینتر که به همین استراکت اشاره می‌کند میسازیم.

به این روش تعریف incomplete data type هم گفته میشه. ما برای ارسال استراکت به توابع از پوینتر استفاده میکنیم.

```
/* incomplete data type to BAM_Handler we use this in all function call's*/
typedef struct BAM_Handler *BAM_Handler_Ptr;
```

شاید بعضی ازممبر ها استفاده نداشته باشه. در حال حاضر مطمئن نیستم. داخل لیست todo مینویسم که در آخر به این موضوع رسیدگی بشه. شاید لازم باشه تعدادی از اونها حذف بشن.

```
/* BAM_Handler is everything we have
 * we pass pointer to instance of this struct to every functions and API's
 * i wonder i actually, do't need some of these data member
 * for ex cur_state, in the end if i dont use any af that i delete it
 *
 * buffer is the led pool 2d array which is the row in the array is BAM_RESOLUTION and
 * the coloumn is NUM_OF_SHFT_REG,
 *
 * max_counter is the maximum delay, that certain state must be spend
 * counter keep track of that,
 * data is the pointer to the arry that must be to send to the shift register chains.
 *
 * */
typedef struct BAM_Handler{

    BAM_STATES_T next_state;
    BAM_STATES_T cur_state;
    uint8_t *buffer;
    uint8_t *data;
    uint8_t max_counter;
    uint8_t counter;
    const bam_state_send_data_fp send_data ;

}BAM_Handler;
```

Led بافر روتعریف میکنیم. سطر ها معرف رزولوشن و ستون‌ها معرف تعداد شیفت رجیستر می‌باشد

```
/*
 * LED_BUFFER holding pwm percentage of each channel in number between 0 - 16
 * becuase we design the 4bit resolution pwm, so we have 4 rows ecach of them has 1bit of the whole pwm value
 * we have 2 coloumn becuase we have 16 channel of pwm, each one for 8 channel or one shift register
 * first row must send to chain in state 0
 * sec row in same manner send to chain in sate 1
 * and goes a round
 * i pick some random number just to show in screen when we saw the operation of program
 * */
uint8_t LED_BUFFER [BAM_RESOLUTION][NUM_OF_SHFT_REG] = {

    { 121, 18 },
    { 12, 127 },
    { 233, 86 },
    { 64, 135 }
};
```

خب رسیدیم به غولش. اینجا بدنه اصلیه استیت ماشین رو طراحی میکنیم. از تکنیکی به نام جدول یا میز حالت استفاده میکنم.

ابتدا یک فانکشن پوینتر داریم.و در ادامه دکورشین استیت ها رومیبینیم که باید مشابه فانکشن پوینتز باشد.
از طرفی بخاطر اینکه از کلمه کلیدی typedef استفاده کردم. پس تبدیل شده به نوع داده ایی حالا میتوانم یه ارایه بسازم که تمام استیت ها را درونش قرار بدم. مهم این است  که ترتیب استیت ها. داخل ارایه باید با enum که بالا تر تعریف کردیم یک شکل و به همان ترتیب باشد

```
 97
 98 /*
 99  * this is function pointer that each state must be look like this*/
100 typedef void (*bam_state_machine_fp) (BAM_Handler_Ptr);
101
102 /* declaration of state function
103  * each satate has own prototype*/
104 void bam_state_func_0 (BAM_Handler_Ptr);
105 void bam_state_func_1 (BAM_Handler_Ptr);
106 void bam_state_func_2 (BAM_Handler_Ptr);
107 void bam_state_func_3 (BAM_Handler_Ptr);
108
109 /* array of function pointer's
110  * we use BAM_TABLE to switch between state's*/
111 const bam_state_machine_fp BAM_TABLE [BAM_STATE_MAX] = {
112     bam_state_func_0,
113     bam_state_func_1,
114     bam_state_func_2,
115     bam_state_func_3
116 };
117
```

تعدادی هم تابع کمکی داریم که به ما کمک میکنند تا از ماشین حالتی که ساخته‌ایم استفاده کنیم.
امیدوارم که به اندازه کافی گویا بوده باشند. داخل کد توضیحات آن‌ها موجود است.

```
/*
 * HELPER FUNCTIONS SECTION
 * */

/* dynamic allocator  for BAM_Handler
 *  its return a pointer to teh new instance of BAM_Handler
 *  it actually return BAM_Handler_Ptr
 *
 * */
BAM_Handler_Ptr bam_init_handler (void);

/*
 * this function must be run in every tick period
 * */
void bam_refresh (BAM_Handler_Ptr);

/* we simply use this for show the result in the screen
 * in the final code we dont use this
 * insted of we put SPI implementation for sending data to shift register chains*/
void bam_print_data (BAM_Handler_Ptr handler);

/*
void bam_print_data (BAM_Handler_Ptr handler);
 *  actually we dont need to this function becuase the state change automaticlly
 *  we implement the changing state in the cocrete state function
 * */
void bam_change_state(BAM_Handler_Ptr handler, BAM_STATES_T next_state);
```

خب رسیدیم به تابع اصلی main :
چیز خاصی نیست تنها ۱۵ بار تابع refresh رو صدا زدم. البته قبل از اون یک object یا instance از handler ساخته‌ام که الان متوجه شدید چقدر زندگی رو اسون و رنگی کرده برامون.
در ضمن تو زندگی واقعی تابع refresh رو باید داخل روتین وقفه مدام صدا بزنیم. در اینجا فقط برای نمایش به تعداد ۱۵ بار فراخوانی شده.

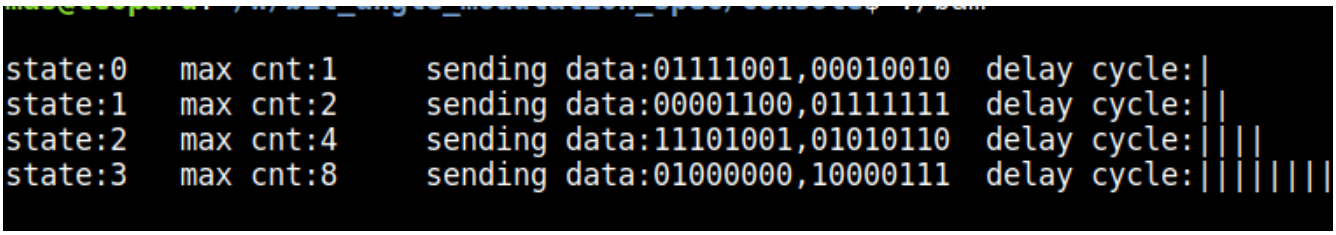
```
 */
    int
main ( int argc, char *argv[] )
{

    BAM_Handler_Ptr handler = bam_init_handler();
/*    BAM_Handler handler = { .next_state = BAM_STATE_BIT_0,
                              .data = LED_BUFFER[BAM_STATE_BIT_0],
                              .counter = 0,
                              .send_data = bam_state_sd };
*/
    int i;
    for (i=0; i<15; i++){
        bam_refresh (handler);
    }

    return EXIT_SUCCESS;
}           /* ----------  end of function main  ---------- */

```

من عکس بخش‌های بعدی رو هم قرار میدم اما خواندن و درک اون رو به خواننده واگذار میکنم.
باشد که جزو نکوکاران باشیم

```
 *  concrete state function
 *  this is the place that we really implement the state's code
 *  everything we have in the program
 *  simply
 *  first of all i send data to the chain
 *  followed by line's of code that  i set the next_state variable's
 *
 *  note that because in the first state we actually spend one cycle period, so we dont check Or
 *  anything else, just jump to next state
 *
 *  but in other state we add a if condition to check send data ,delay or just jump over the state
 *  i hope its make sense :))
 *
 * */
void bam_state_func_0 (BAM_Handler_Ptr handler){
    handler->send_data(handler);

    handler->next_state = BAM_STATE_BIT_1;
    handler->data = LED_BUFFER[BAM_STATE_BIT_1];
    handler->counter = 0;
    handler->max_counter = BAM_STATE_1_MAX_COUNTER;
}
```

```
void bam_state_func_1 (BAM_Handler_Ptr handler){
    if(!handler->counter){
        handler->send_data(handler);

    }else{
        handler->next_state = BAM_STATE_BIT_2;
        handler->data = LED_BUFFER[BAM_STATE_BIT_2];
        handler->counter = 0;
        handler->max_counter = BAM_STATE_2_MAX_COUNTER;
        return;
    }

    handler->counter = handler->counter + 1;
}
void bam_state_func_2 (BAM_Handler_Ptr handler){
    if(!handler->counter){
        handler->send_data(handler);

    }else if (handler->counter >= handler->max_counter){
        handler->next_state = BAM_STATE_BIT_3;
        handler->data = LED_BUFFER[BAM_STATE_BIT_3];
        handler->counter = 0;
        handler->max_counter = BAM_STATE_3_MAX_COUNTER;
        return;
    }

    handler->counter = handler->counter + 1;
}
```

حالا وقتشه خروجی برنامه رو ببینیم .




در قسمت بعدی آموزش از این تکنیک داخل یک پروژه واقعی استفاده میکنم.

این آموزش بطور آزاد منشر می شود. شما میتوانید آن را ویرایش و بازنشر کنید.

این آموزش برای اولین بار در گروه چیپکده انتشار می‌یابد.

**مسعود ثمرین**
تابستان سال ۹۶