bool FMstate, audioState;

bool signalSource. 0 for FM
1 for audio

发 main. VUART Resync.

初始化, ~~[illegible]~~

~~UART [illegible] → buffer.~~

调用UART处理,

获取指令.

指令 → 曲谱管理

曲谱记录缓冲

有可以取的音符

→ 音符 → FM

→ 音符 → 声音管理&合成

发 Systick.

+ 曲谱记数器更新

+ 声音合成更新 → PWM;

+ VUART重同步

收 main.

初始化, 等待重同步信号

调用UART处理,

获取指令

指令 → FM接收 检查有无新的字节.

指令 & FM信号 →

~~曲谱缓冲 或 记录.~~

音符对齐缓冲

UART指令 FM字节接收

指令 字节

音符对齐缓冲

音符.

曲谱管理

曲谱记录缓冲

音符

声音管理&合成.

Systick.

+曲谱 ~

+

废弃 midi 音符中的 ~~音~~，用于对齐

trimmed midi 8bit 音符，8bit 时长（可延时，至多 2Byte）。

用面向） uint8_t ~~uint~~ process UART ~~MM~~ Input (void)

UART处理： ~~UART Char Arrival ( )~~ 循环。

从 UART 2读有字符到达 → 缓冲区，只检查一次 ~~...~~。

如果 buffer 将溢出，向 UART2 输出错误 & 清空buffer。

如果末尾是 \r \n，~~进入指令检查~~：清除 \r \n，进入 analyzeCommand ~~Receiver~~ ( )，返回 ~~返~~ 返回的指令，否则返回 0。

uint8_t analyze Command Receiver (~~ch~~ char*)

/Transmitter (char*)

	发		收
指令集	play %d	播放%d号曲子。	~~record on~~
	pause	—	~~record off~~
	continue.	—	record %d ~
	FM ~~on~~ %d	0 for off 1 for on.	play 和 record. ~~play record~~.
	audio %d	~	play 和 FM.

	发	收
返回	0b 10 000 xxx	0b 11 000 xxx
	0b 10 001 xxx	0b 11 001 xxx
	0b 10 010 xxx	0b 11 010 xxx
	0b 10 011 xxx	0b 11 011 xxx
	0b 10 100 xxx	0b 11 100 xxx

→ MIDI获取时防下F.

音符: typedef {

bool noteResyncFlag = false.

uint8_t pitch

uint16_t ~~duration~~ delay

uint8_t intensity

} note;

~~音符对齐缓冲: bool~~

~~音符~~清理 void clearNote (note* ~~Note~~ x); ,全写成0.

~~音符对齐缓冲~~: bool noteAligning (note* x, uint8_t Byte)

返回对齐是否完成;读到下F清掉该音符,置标记

依次写入 pitch, intensity 和 duration.

delay ~~duration~~ 0xxxxxxx → 直接写, 返回true. 否则返回false,

下次返回 true, 下次Byte左移一位

~~曲谱管理: void switchToScore (uint8_t)~~ switchToScore (Score* newScore)

typedef {

~~note*~~ ~~notelist~~ note* noteList.

uint32_t noteNum

uint32_t ~~flag~~ Cursor; ~~FM Cursor~~

bool paused;

} musicScore;

曲谱管理：void ~~initialize~~ initScore (musicScore*, note*, uint32_t);

void processScoreCommand (musicScore*, uint32_t command);

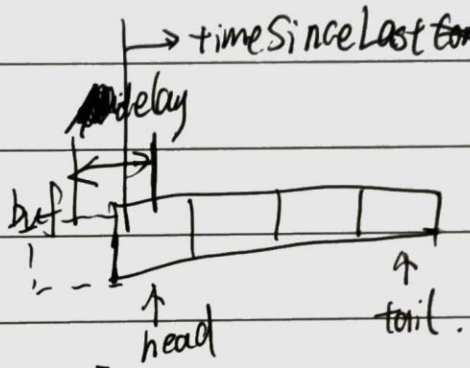
bool noteAvailForAudio (m~S*) 有已到时的 note.

bool noteAvailforFM (m~S*) 之后 1024ms 内有 note.

void getNoteforAudio (m~S*, note*)

void ___ FM 同上.

bool isNoteLeftinScore (m~S*)



曲谱缓冲区:

~~typedef~~ {

note buffer[32]; 循环队列

int8_t head, tail.

// ~~int8_t FM cursor~~.

uint16_t timeSinceLastCommand; // 单位 ms

} ~~note~~ scoreBuffer.

void clearBuf(sc_Buf*);

void update ScoreCounter (s~Buf*): 就管 [illegible]

uint8_t getBufferRemainingSize (~)

~~bool isScBuf isBufferEmpty (s~Buf*)~~

bool isBufferFull (~*).

bool isNoteAvail (scBuf*); 请[illegible]的

然后返回

void getNote (Buf*, Note*); 不~)

更新 timeSinceLastCmd.

曲谱录制及信源选择.

void recordToScore (Sc*, note*).

bool isScoreFull (Score*

void processScoreManage (Score*, uint8_t cmd)

bool isAbleToWriteNoteToBuffe (*tx,).

void Write NotetoBuf (*tx, *note).

声音合成: 简版: , musicState*.

void noteCommand (note*,) 开关note.

uint16_t getSignal (uint32_t time , musicState*)