

H_Ard_NG50_Base

Generatore musicale MIDI per Arduino Mega / Due

- Avvio e controllo
- Tempo, tick e BPM
- Struttura di un brano
- Pattern, esecutori ed effetti
- Compositore automatico

Il manuale descrive il comportamento del software così come è stato verificato in simulazione su PC. Le parti che dipendono dall'hardware reale (display, rete, tasti, tempi reali) vanno verificate sulla scheda.

Indice

Introduzione.....	4
Scopo del manuale.....	4
Architettura in breve.....	4
Convenzioni del manuale.....	4
Stato delle verifiche.....	4
1 Avvio, start/stop e tasti.....	5
1.1 Stato di partenza.....	5
1.2 Start e stop.....	5
1.3 Modalità dei tasti.....	5
1.4 Protezione all'accensione.....	6
1.5 Comandi dalla seriale del nucleo.....	6
2 Funzioni escludibili (config.h).....	7
3 Liste di esecuzione e memoria.....	8
3.1 playList ed endList.....	8
3.2 Dimensioni.....	8
3.3 Cosa succede quando una lista è piena.....	8
3.4 Tarare le dimensioni.....	8
3.5 Note sulla libreria PlayEndLib.....	8
4 Diagnostica con la seriale (debug).....	9
4.1 Attivazione.....	9
4.2 Messaggi.....	9
4.3 Come leggere "Tick max us".....	9
5 Il tempo: tick, BPM, durate.....	10
5.1 Il tick.....	10
5.2 BPM e battute.....	10
5.3 Posizioni e durate.....	10
5.4 Precisione.....	10
6 Com'è fatto un brano.....	11
6.1 Schema di base.....	11
6.2 Variabili principali.....	11
6.3 Dividere il brano in sezioni.....	11
7 Parametri di un brano.....	12
7.1 Tonalità e scala.....	12
7.2 Volume.....	13
7.3 Canali.....	13
7.4 Control change casuali.....	13
7.5 Fine del brano.....	13
8 Silenzi (note vuote).....	14

8.1 Silenzi nei pattern.....	14
8.2 Esempio.....	14
8.3 Comportamento delle pause.....	14
8.4 Silenzi generati.....	14
8.5 Note tecniche.....	15
9 Accordi.....	16
9.1 Accordi dal pattern.....	16
9.2 Accordi da vettore.....	16
10 Pattern: scriverli a mano e generarli.....	18
10.1 Il vettore basePat.....	18
10.2 Leggere e scrivere.....	18
10.3 Generatori.....	18
10.4 Variazioni su un pattern esistente.....	19
11 Esecutori ed effetti.....	20
11.1 Parametri comuni.....	20
11.2 Esecutori.....	20
11.3 Effetti.....	20
11.4 Moduli liberi.....	21
12 Il compositore automatico (complexp).....	22
12.1 La struttura complexp.....	22
12.2 Effetto sul brano.....	22
12.3 Sezioni e tracce.....	22
13 Esempio completo di brano.....	24
13.1 Il codice.....	24
13.2 Come funziona.....	24
13.3 Risultato.....	24
13.4 Ricette rapide.....	25
14 Scegliere il brano attivo.....	26
15 Cose da sapere.....	27
Appendice A Risoluzione dei problemi.....	28
Appendice B Glossario.....	29

Introduzione

Scopo del manuale

Questo manuale spiega come si usa e come si programma `H_Ard_NG50_Base`, uno sketch Arduino che genera musica e la invia come messaggi MIDI. Descrive l'avvio e i comandi, il modo in cui il tempo viene scandito, la struttura di un brano, gli strumenti per scrivere o generare pattern e gli esecutori che trasformano i pattern in note.

Il manuale è pensato per chi vuole scrivere un proprio brano o modificare quelli esistenti: si presuppone una conoscenza di base dell'IDE Arduino e del linguaggio C/C++.

Architettura in breve

Il funzionamento ruota attorno a un unico concetto: il **tick**, l'unità minima di tempo. A ogni tick il sistema esegue sempre la stessa catena di operazioni:

Passo	Cosa succede
1. Tick	Lo scheduler (<code>micros()</code> , <code>Scheduler.h</code>) fa scattare il tick con l'intervallo calcolato dal BPM.
2. <code>carica()</code>	Incrementa <code>contick</code> e chiama la funzione del brano attivo (capitolo 14).
3. Brano	Decide cosa suonare in base a <code>contick</code> : all'inizio prepara tempo, tonalità e pattern, poi a ogni battuta chiama gli esecutori.
4. Esecutori	Leggono i pattern in <code>basePat</code> e mettono in coda le note con <code>playAdd(nota, midichan, volume, durata, inizio)</code> .
5. <code>playList</code>	Contiene le note in attesa: quando arriva il tick di inizio la nota viene suonata.
6. <code>endList</code>	Contiene le note in corso: al tick <code>inizio + durata</code> la nota viene spenta.

Tutte le posizioni e le durate sono espresse in tick. I pattern non contengono note assolute ma **gradi della scala**: per questo lo stesso pattern si trasporta automaticamente quando si cambia tonalità.

Convenzioni del manuale

Notazione	Significato
<code>nomeFunzione()</code>	nomi di funzioni, variabili, file e macro del codice
Blocco scuro	esempio di codice da copiare nello sketch
NB	nota importante o comportamento non ovvio
ATTENZIONE	situazione che può causare malfunzionamenti o risultati inattesi
SUGGERIMENTO	consiglio pratico
(vedi 7.5)	rimando a un capitolo o paragrafo del manuale

Stato delle verifiche

Tutto ciò che è descritto è stato verificato sul codice, simulandolo su PC con l'hardware simulato (tick, note, silenzi, accordi, tasti all'avvio, avvio automatico). Display, rete, sensori e tempi reali non sono stati provati: vanno verificati sulla scheda.

1 Avvio, start/stop e tasti

Questo capitolo descrive come il sistema si accende, come si avvia e si ferma un brano e quali comandi sono disponibili dai tasti o dalla seriale.

1.1 Stato di partenza

La variabile `boolean acceso` (in `H_Ard_NG50_Base.ino`) stabilisce cosa succede all'accensione:

Valore	Comportamento
<code>true</code>	il brano parte da solo alla fine di <code>setup()</code>
<code>false</code>	il sistema resta fermo finché non si preme start

In ogni caso `setup()` porta prima il sistema nello stato “spento” chiamando `fermaTutto()` (come faceva il vecchio `OnOff()` in `setup`). Solo alla fine, se `acceso` era `true`, chiama `OnOff()`, cioè start. In questo modo il sistema parte sempre da uno stato pulito, senza note appese.

1.2 Start e stop

`OnOff()` inverte lo stato: se il brano è fermo lo avvia, se è in esecuzione lo ferma.

Azione	Cosa fa
Start	se <code>contick == 0</code> chiama <code>carica()</code> per preparare il brano, invia <code>MIDISTART</code> e attiva il tick.
Stop	<code>fermaTutto()</code> : ferma i tick e spegne tutte le note.

1.3 Modalità dei tasti

La modalità si sceglie con `KEYS_MODE` in `config.h` (vedi capitolo 2).

KEYS_FULL – tastiera completa

I tasti sono letti da `ioi.readKey()`, tastiera tipo 3 “4 KEY”.

Tasto	Sigla	Funzione
4	E	fine differita: <code>diffstop = true</code> (vedi 7.5)
5	F	start/stop
6	select	start/stop
8	N	non usato
9	C	attiva/disattiva i control change casuali (vedi 7.4)
10	A/M	non usato
11	X	reset valori: la prima pressione arma, la seconda entro 3 secondi azzerà tutto

KEYS_ONOFF – tasto singolo

Un solo tasto sul pin `PIN_ONOFF` (`config.h`, default 7), collegato verso massa con pull-up interno: funziona da start/stop.

ATTENZIONE Adattare `PIN_ONOFF` al cablaggio reale della scheda.

KEYS_NONE – nessun tasto

Nessun tasto collegato: il brano si avvia solo con `accesso = true` oppure con un comando dalla seriale del nucleo.

1.4 Protezione all'accensione

Nei primi **500 ms** dall'avvio i tasti vengono ignorati: il valore letto serve solo come stato di partenza. Così un tasto “bloccato” o un disturbo elettrico all'accensione non fanno partire il brano per errore. In modalità debug ogni pressione stampa **Tasto n** sulla seriale.

1.5 Comandi dalla seriale del nucleo

Con `USE_NETWORK` attivo il nucleo (`Serial1`) può inviare comandi di testo:

Comando	Effetto
<code>start</code>	avvia il brano
<code>stop</code>	ferma il brano
<code>dstop</code>	fine differita (vedi 7.5)

2 Funzioni escludibili (config.h)

Alcune parti del sistema possono essere escluse dalla compilazione. È utile per risparmiare memoria e, soprattutto, tempo di calcolo: su schede lente il display o la rete possono rubare tempo al tick.

In `config.h` ogni opzione vale `1` (attiva) o `0` (esclusa). Un'opzione esclusa non occupa né memoria né tempo: le chiamate spariscono, argomenti compresi, quindi non vengono costruite nemmeno le `String` passate come parametro.

Opzione	Descrizione
<code>USE_DISPLAY</code>	Display LCD 5110 (libreria <code>I0Int</code>). Le chiamate sono le macro <code>LCD_print</code> , <code>LCD_printNumI</code> , ... Il display si aggiorna solo se è cambiato qualcosa e se resta tempo prima del prossimo tick.
<code>USE_NETWORK</code>	Socket / OSC / I2C verso display esterno e moduli (ExtMem), seriale col nucleo, memoria esterna. Anche se compilata, i messaggi partono solo se <code>FunStatus[4]</code> non è ' ' (Mega: 'S' = socket).
<code>USE_SENSORI</code>	Sensori di luce, suono e movimento (ExtSensori). Nel codice attuale sono solo inizializzati.
<code>KEYS_MODE</code>	<code>KEYS_NONE</code> / <code>KEYS_ONOFF</code> / <code>KEYS_FULL</code> (vedi 1.3).

Le opzioni sono protette da `#ifndef`: oltre che nel file `config.h` si possono forzare dall'esterno con `-D` in compilazione.

SUGGERIMENTO Se la musica non sta al passo, come prima prova impostare `USE_DISPLAY 0` e `USE_NETWORK 0` (vedi anche capitolo 4).

3 Liste di esecuzione e memoria

Le note non vengono suonate nel momento in cui un esecutore le genera: vengono messe in coda e suonate quando arriva il loro tick. Per farlo il sistema usa due liste.

3.1 `playList` ed `endList`

Lista	Contenuto	Occupazione
<code>playList</code>	note da suonare, in attesa del loro tick di inizio	8 byte per nota
<code>endList</code>	note in corso, da spegnere al termine della durata	4 byte per nota

Sono array statici della libreria `PlayEndLib`: lo spazio è sempre riservato, anche quando le liste sono vuote.

3.2 Dimensioni

Le dimensioni si impostano in `config.h` con `PLAYELE` (`playList`) ed `ENDELE` (`endList`):

Scheda	PLAYELE	ENDELE
Mega	96	32
Due	256	128

3.3 Cosa succede quando una lista è piena

- Se `playList` è piena, la nota in più viene scartata e compare `'0'` (overflow).
- Se `endList` è piena, la nota appena suonata viene spenta subito, per non lasciare note appese.

3.4 Tarare le dimensioni

In debug, allo stop, viene stampato `Max play/end: a/b`, cioè i picchi raggiunti dalle due liste. Valori misurati in simulazione:

Caso	<code>playList</code>	<code>endList</code>
Brani di prova	24	3
Compositore completo (tutte le tracce, base 32)	64	19

SUGGERIMENTO Impostare le dimensioni poco sopra il picco misurato: la RAM risparmiata resta disponibile per i pattern (vedi capitolo 15).

3.5 Note sulla libreria `PlayEndLib`

- La libreria è separata (`librariesDev/PlayEndLib`). Il parametro del suo template si chiama `N` e non `PLAYELE`, perché `PLAYELE` ed `ENDELE` sono macro del progetto.
- Nelle firme di funzione si usa `PlayListT` (typedef in `config.h`), perché l'IDE Arduino non genera i prototipi se tra `<` `>` compare un numero.
- `deleteCurrent()` non mantiene l'ordine (l'ultimo elemento prende il posto di quello cancellato); per mantenerlo esiste `deleteCurrentOrdered()`. Il progetto non ne ha bisogno.

4 Diagnostica con la seriale (debug)

La modalità debug serve a vedere sulla seriale cosa genera il sistema e a misurare se il tempo di calcolo è sufficiente.

4.1 Attivazione

Impostare `boolean debug = true;` in `H_Ard_NG50_Base.ino` e aprire il monitor seriale a **9600** baud.

ATTENZIONE In debug i messaggi MIDI escono come testo ("`Cmd: 144 - Nota 60 - Vel. 100`") e NON come byte MIDI: lo strumento collegato non suona. La modalità serve solo a controllare cosa viene generato.

4.2 Messaggi

Messaggio	Significato
Status: 1 / Status: 0	start / stop
Max play/end: a/b	picchi delle liste dall'ultimo avvio (stampato allo stop)
Tick max us: n	durata del tick più lungo, in microsecondi (stampato allo stop)
Tasto n	tasto premuto (solo KEYS_FULL)
Spengo TUTTE le note	stop
Invio dati generali	aggiornamento dello stato verso l'esterno (solo con <code>accesso = true</code>)

4.3 Come leggere "Tick max us"

L'intervallo disponibile per un tick è:

```
intervallo_tick_us = 60.000.000 / (BPM x 16)
```

Tick max us indica quanto è durato il tick più pesante. Se si avvicina all'intervallo o lo supera, il sistema non riesce a stare al passo e le note arrivano in ritardo. Rimedi, in ordine di semplicità:

- escludere display e rete (capitolo 2);
- alleggerire il brano (meno tracce o effetti per battuta);
- abbassare il BPM.

Esempio: a 120 BPM il tick dura 31.250 µs; un **Tick max us** di 28.000 indica che il margine è quasi esaurito.

5 Il tempo: tick, BPM, durate

5.1 Il tick

Il brano è scandito dal tick. La durata di un tick è $60 / (\text{BPM} \times 16)$ secondi (`beatdiv = 16`): una battuta di riferimento è divisa in 16 tick.

BPM	Durata tick	Battuta da 16 tick
20	187,5 ms	3 s
30	125 ms	2 s
60	62,5 ms	1 s
120	31,25 ms	0,5 s

5.2 BPM e battute

- Una battuta di 16 tick dura $60 / \text{BPM}$ secondi: con `base1 = 16` il BPM coincide quindi con le battute al minuto.
- Con `base1 = 32` la battuta dura il doppio.
- Il BPM è `PotAverage[9]` (10..240). Si può cambiare durante l'esecuzione (`PotAverage[9] = 40;`): `loop()` aggiorna l'intervallo e il cambio ha effetto subito.

5.3 Posizioni e durate

Ogni posizione e ogni durata sono in tick. Una nota con inizio `i` e durata `d` parte al tick `i` e si spegne al tick `i + d`.

Esempio: una durata di 2 tick a 30 BPM corrisponde a $2 \times 125 \text{ ms} = 0,25 \text{ s}$.

5.4 Precisione

Il tick è regolato con `micros()` (`Scheduler.h`) e non accumula deriva: se un giro è in ritardo, il successivo recupera. Se si perde più di un tick di fila, lo scheduler riparte dall'istante attuale invece di eseguire una raffica di tick arretrati.

6 Com'è fatto un brano

Un brano è una normale funzione C++ chiamata a ogni tick da `carica()`, subito dopo `contick++`. La funzione non “suona” direttamente: a ogni chiamata decide se è il momento di preparare qualcosa e mette in coda le note.

6.1 Schema di base

```
if (contick == 0) { ... }           // inizializzazione, una sola volta: tempo, tonalità, pattern
if (contick % base1 == 0) { ... }  // a ogni inizio battuta: si suonano i pattern
```

Il primo blocco viene eseguito una sola volta, al primo tick; il secondo all'inizio di ogni battuta. Tra una battuta e l'altra la funzione viene comunque chiamata, ma non fa nulla: le note già in coda vengono suonate dalle liste.

6.2 Variabili principali

Variabile	Significato
<code>contick</code>	contatore dei tick dall'inizio. È unsigned: su AVR (Mega) è a 16 bit e riparte da 0 dopo 65535; sulla Due è a 32 bit.
<code>base1</code>	tick per battuta (16 o 32 negli esempi).
<code>tickDivider(n, base1)</code>	numero della battuta, da 0 a n-1, dentro un ciclo di n battute.
<code>DatiVideo.*</code>	<code>baselen</code> , <code>songlen</code> , <code>bitfull</code> , <code>flagmodifica</code> : servono solo al display e all'esterno (tick per battuta, battute del brano, battuta attuale, “c'è da aggiornare”).

NB Usare potenze di 2 per `base1` e per il numero di battute: così, quando `contick` riparte da 0, il ciclo del brano non si spezza a metà.

6.3 Dividere il brano in sezioni

`tickDivider()` permette di organizzare il brano in sezioni che si ripetono ciclicamente:

```
bitarcf = tickDivider(64, base1); // 0..63, poi riparte
if (bitarcf <= 3) { /* intro */ } else if (bitarcf <= 19) { /* strofa */ } ...
```

In questo esempio il brano dura 64 battute: le prime 4 sono l'intro, le battute da 4 a 19 la strofa, e così via.

7 Parametri di un brano

I parametri principali di un brano sono contenuti nell'array **PotAverage** (le variabili "pot", che nelle versioni con potenziometri corrispondono alle manopole).

Indice	Significato
PotAverage[1]	canale dei control change (7.4)
PotAverage[2]	riga dei ritmi (0..7), per il motore ritmico
PotAverage[3]	tonalità e scala (7.1)
PotAverage[7]	prossima tonalità (solo visualizzazione)
PotAverage[8]	volume base, 0..127 (default 100) (7.2)
PotAverage[9]	BPM, 10..240 (capitolo 5)
[0], [4], [5]	non usati

7.1 Tonalità e scala

Tonalità e scala sono codificate in un solo numero e attivate con **setModeKeySong()**:

```
PotAverage[3] = riga * 12 + nota;      setModeKeySong(PotValue(3));
```

nota è la tonica:

0	1	2	3	4	5	6	7	8	9	10	11
Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si

riga è il tipo di scala (tabella **scaleMod** in **config.h**):

Riga	Scala	Riga	Scala
0	maggiore (ionico)	10	minore melodica
1	minore	11	minore armonica
2	dorico	12	maggiore armonica
3	frigio	13	orientale
4	lidio	14	ungherese maggiore
5	misolidio	15	enigmatica
6	eolio	16	pentatonica maggiore
7	locrio	17	diminuita
8	napoletana	18	toni interi
9	napoletana minore	19	tutti i semitoni

Valore	Calcolo	Risultato
0	$0 \times 12 + 0$	Do maggiore
12	$1 \times 12 + 0$	Do minore
14	$1 \times 12 + 2$	Re minore
24	$2 \times 12 + 0$	Do dorico

Dopo **setModeKeySong()** sono pronti **keyArray[]** (le note della scala) e **note4key** (quante sono).

NB I pattern memorizzano gradi della scala, non note: cambiando tonalità si trasportano da soli.

7.2 Volume

PotAverage[8] è il volume base. Per rendere l'esecuzione meno meccanica, gli esecutori lo variano a caso del 20% una volta ogni 3 note (**volumePlus(volume, 20, 3)**).

Le versioni “complete” di alcuni esecutori accettano anche **startvolume** e **diffvolume**: il primo è il volume iniziale, il secondo l'incremento per nota, positivo per un crescendo e negativo per un diminuendo.

```
moduloBase(0, 16, 0, 0, 1, 40, 4);    // parte da 40 e sale di 4 a ogni nota
```

7.3 Canali

midichan, ultimo parametro di quasi tutti gli esecutori, è il canale MIDI 0..15 (0 = canale 1) e anche l'etichetta (tag) della traccia. Una nota si aggancia a una già in corso solo se hanno lo stesso tag e la stessa altezza.

SUGGERIMENTO Usare canali diversi per tracce diverse: oltre a poterle assegnare a strumenti diversi, si evita che note uguali di tracce diverse si fondano.

7.4 Control change casuali

Si attivano con il tasto 9 (**FunStatus[1] = 'C'**). Inviano 8 controller (0..7) con valori casuali, ogni **60/BPM** secondi, sul canale **PotAverage[1]**.

7.5 Fine del brano

Per terminare un brano al momento giusto invece di interromperlo bruscamente si usa la fine differita: **diffstop = true** (oppure il tasto E, oppure il comando **dstop** dal nucleo). **carica()** ferma il brano al tick successivo.

NB **carica()** contiene due righe di stop differito; la prima scatta sempre per prima, quindi la seconda (fine a multipli di 1024 tick) non viene mai raggiunta. Se si vuole la fine a multipli di 1024 tick, commentare la prima.

8 Silenzi (note vuote)

Il silenzio è parte della musica quanto le note. In `H_Ard_NG50_Base` un silenzio si ottiene semplicemente non mettendo nessuna nota in un certo intervallo; dentro i pattern lo si indica con un valore speciale.

8.1 Silenzi nei pattern

In `basePat` un silenzio è un elemento con intervallo **255** e ottava **99**. Si scrive in due modi equivalenti:

```
nota2int(255, i);           // con il valore MIDI speciale 255
int2int(255, 99, i);        // con grado e ottava
```

Gli esecutori (`moduloBase` e gli altri) saltano questi elementi: nessuna nota viene inserita e quindi in quel tempo c'è silenzio.

8.2 Esempio

```
nota2int(60, 0); nota2int(60, 1);    // Do Do
nota2int(255, 2); nota2int(255, 3);  // silenzio di 2 tick
nota2int(62, 4);                     // Re
int2int(255, 99, 5);                 // silenzio di 1 tick
nota2int(64, 6); nota2int(64, 7);    // Mi Mi
moduloBase(0, 8, 1);                 // suona il pattern sul canale 1
```

Risultato: Do dal tick 0 al 2, Re dal 4 al 5, Mi dal 6 all'8. Le due caselle consecutive con la stessa nota diventano una sola nota lunga 2 tick.

8.3 Comportamento delle pause

- Una pausa tra due note uguali le tiene separate (`Do _ Do` = due note); senza pausa si fondono (`Do Do Do` = una nota lunga 3).
- Una pausa all'inizio ritarda l'attacco; una pausa in fondo lascia silenzio fino alla fine del pattern.
- Se non si programma nulla in un intervallo c'è già silenzio: una nota con durata più corta della sua casella lascia muto il resto.
- Per accordi e arpeggi: `fastloadPlay(255, durata, inizio, ...)` è una pausa.

8.4 Silenzi generati

Per inserire un silenzio casuale in un pattern già pieno:

```
fxRandomPauseSingle(starti, patlen, ltick, startpos)
fxRandomPauseSingle(0, 8, 2, 3);           // 2 tick di silenzio a partire dal tick 3
```

Parametro	Valore speciale
<code>ltick</code>	0 = lunghezza casuale (2 o 4)
<code>startpos</code>	-1 = posizione casuale

Alcuni generatori (`baseFullControl`, `baseFibonacci`) hanno il parametro `modoblank`:

modobLank	Effetto
0	casuale
1	nessun silenzio
2	mezza nota muta
3	tutta la nota muta

8.5 Note tecniche

- Le pause non occupano posto in `playList`: la loro presenza non rallenta né riempie la lista.
- Le funzioni che controllano le “note ancora in play” di un canale (`contaPlayTick`, usata da `moduloArpGlass` e `moduloMedia`) non contano le pause: possono quindi ripartire un attimo prima, quando resta solo silenzio.
- `fxRandomPauseMulti` è segnata “NON FUNZIONA” nei commenti del codice e non è stata modificata.

9 Accordi

Gli accordi si possono ottenere in due modi: trasformando in accordo ogni nota di un pattern, oppure costruendo un accordo in un vettore e suonandolo con una delle modalità disponibili.

9.1 Accordi dal pattern

Alcuni esecutori accettano **modoaccordo** e **tipoaccordo**: ogni nota del pattern diventa un accordo costruito su quella nota.

modoaccordo	Significato
0	nota singola (nessun accordo)
1, 2, 3 ...	1 = scala maggiore, 2 = minore, 3 = dorico ... (riga di scaleMod = modoaccordo - 1, vedi 7.1)

tipoaccordo	Accordo	tipoaccordo	Accordo
0	casuale	7	nona aumentata
1	Maj/Min (triade)	8	nona minore
2	settima maggiore	9	undicesima
3	settima	10	undicesima aumentata
4	nona aggiunta	11	tredicesima
5	maggiore con nona	12	sesta
6	nona		

Esempio: ogni nota diventa una triade maggiore sul canale 3.

```
moduloBase(0, 16, 1, 1, 3); // Do -> 60,64,67 Re -> 62,66,69
```

NB La triade è costruita sulla nota (Re maggiore = Re Fa# La) e non è per forza dentro la tonalità del brano.

9.2 Accordi da vettore

accordi[] contiene le note assolute dell'accordo (0..11), terminate da **99**. Si riempie con **GeneraAccordi()** e si suona con **playAccordiNoBasePat()**:

```
GeneraAccordi(tonalita, tipo, 1, accordi, riga); // riempie accordi[]  
playAccordiNoBasePat(starti, base, ottava, notastart, spostata, modo, playorpat, midichan);
```

Il parametro **modo** stabilisce come viene eseguito l'accordo:

modo	Esecuzione	modo	Esecuzione
1	note sovrapposte per tutta la durata	8	note a lunghezza casuale
2	note una dopo l'altra, durata divisa	9	arpeggio
3	sovrapposizione parziale	10	basso continuo
4	sovrapposizione parziale multipla	11	basso continuo in terzine
5	un colpo di mezza durata	12	pattern dispari
6	due colpi	13	casuale per terze/quinte

modo	Esecuzione	modo	Esecuzione
7	blocchi casuali di 1/8	14	arpeggio 0-1-0-2

Con **modo = 0** la scelta è casuale tra 1 e 2.

- **playorpat**: **0** suona subito, **1** genera un pattern invece di suonare (utile per rielaborarlo con gli effetti).
- Se **accordi[]** è vuoto la funzione non fa nulla.

10 Pattern: scriverli a mano e generarli

Il pattern è la materia prima di ogni brano: una sequenza di caselle, una per tick, ognuna con un grado della scala e un'ottava (oppure un silenzio). I pattern si possono scrivere a mano, nota per nota, o farli generare da funzioni che applicano regole diverse.

10.1 Il vettore basePat

`basePat[i]` contiene, per ogni tick, un grado della scala (intervallo) e un'ottava. La dimensione è **BASEPATELE**: 512 su Mega e Due, 256 sulle altre schede.

NB Il compositore automatico usa zone di lavoro alte del vettore (per esempio da 192 in su): nei brani scritti a mano conviene usare indici bassi.

10.2 Leggere e scrivere

Funzione	Descrizione
<code>nota2int(nota_midi, i)</code>	scrive la nota MIDI nel pattern, portandola sul grado più vicino della scala
<code>int2int(grado, ottava, i)</code>	scrive direttamente grado e ottava (255, 99 = silenzio)
<code>int2nota(i)</code>	restituisce la nota MIDI (255 se silenzio)
<code>duplicaPattern(starti, desti, n)</code>	copia n elementi
<code>duplicaPatternInv(starti, desti, n)</code>	copia n elementi in ordine inverso
<code>sumint / spostaPattern(starti, numnote, spostamento)</code>	sposta le note di un certo numero di gradi

10.3 Generatori

Tutti i generatori riempiono il pattern a partire da `vettstart` per `base` tick. `ottavastart` e `ottavaend` delimitano l'intervallo di ottave usato.

Funzione	Descrizione
<code>creabasePat(fullp, base, npattern, ottavastart, ottavaend)</code>	pattern casuale di <code>base</code> tick; <code>fullp = true</code> lo costruisce; <code>npattern</code> = quanti pattern produce (per farne uno solo oltre al base, indicare 2)
<code>baseFullControl(vettstart, base, mmin, mmax, normalizza, modoblank, ottavastart, ottavaend)</code>	cambi di nota a intervalli casuali: <code>mmin/mmax</code> = lunghezza minima/massima (0 = casuale), <code>normalizza</code> = arrotonda a multipli, <code>modoblank</code> = silenzi (8.4)
<code>baseFixNote(numnote, base, vettstart, vettdest, tipocalcolo, ottavastart, ottavaend)</code>	<code>numnote</code> note; tipocalcolo : 0 tutto casuale, 1 copia le note e cambia le durate, 2 copia le durate e cambia le note
<code>baseRegularStep(vettstart, base, lenstep, ottavastart, ottavaend)</code>	note a passo regolare (<code>lenstep</code> 0 = casuale)
<code>basePesato(vettstart, base, lenstep, ottavastart, ottavaend)</code>	scelte pesate sui parametri di <code>complexp</code> (capitolo 12)
<code>baseFibonacci(vettstart, base, modoblank, addval, ottavastart, ottavaend)</code>	costruito sulla serie di Fibonacci
<code>baseFromAccordi(vettstart, base, lenstep, ottavastart, startacc)</code>	usa solo le note degli accordi

10.4 Variazioni su un pattern esistente

Per evitare ripetizioni identiche, un pattern già scritto può essere variato leggermente:

Funzione	Descrizione
<code>notePlus(numnote, vettstart)</code>	variazione sul pattern esistente
<code>noteMinus(numnote, vettstart)</code>	variazione sul pattern esistente
<code>moduloMinDiff(starti, desti, base)</code>	piccola modifica sulla nota più lunga

11 Esecutori ed effetti

Gli esecutori sono le funzioni che leggono un pattern e mettono le note in **playList**. Gli effetti sono esecutori che, mentre leggono il pattern, lo trasformano: lo dilatano, lo invertono, lo arpeggiano. Lo stesso pattern può quindi essere suonato in modi diversi su tracce diverse.

11.1 Parametri comuni

Parametro	Significato
sti / starti	dove inizia il pattern in basePat
patlen / notepattern	quanti tick leggere
spostanota	alza la nota di quel numero di gradi nella scala (0 = invariata)
ltick	lunghezza in tick usata dall'effetto
midichan	canale MIDI / tag della traccia (7.3)

11.2 Esecutori

Funzione	Descrizione
moduloBase(sti, notepattern, midichan)	suona il pattern; esistono versioni con modoaccordo , tipoaccordo , startvolume , diffvolume
moduloBasePlus(...)	come moduloBase , ma aggancia la prima nota all'ultima precedente
moduloBaseDeviant(starti, notepattern, varlen, modoaccordo, tipoaccordo, midichan)	accorpa le note; varlen : 0 accorpa, 1 allunga fino in fondo, 2 casuale
moduloBaseSposta(sti, notepattern, spostanota, midichan)	come moduloBase , spostando di gradi
moduloSingle / moduloRangeSingle / moduloRangeNota	note singole o distribuite sui canali

11.3 Effetti

Funzione	Descrizione
moduloEspandi(sti, patlen, ltick, spostanota, midichan)	dilata il tempo: ogni tick dura ltick
moduloContrai(sti, patlen, divisore, spostanota, midichan)	una nota ogni divisore tick
moduloBeatRev(sti, patlen, offset, spostanota, midichan)	la battuta al contrario (offset alza/abbassa)
moduloCounterpUniversal(sti, patlen, ltick, spostanota, midichan)	contrappunto: sfasa di ltick
moduloDivBattuta(starti, patlen, ltick, spostanota, midichan)	normalizza le note a lunghezza ltick
moduloCambioOttava(starti, ltick, midichan)	cambia a caso l'ottava
moduloTerze(sti, patlen, spostanota, midichan)	sposta di terze/quinte (0 = casuale $\pm 2/4$)
moduloTriads(starti, patlen, ltick, midichan)	triplica le note

Funzione	Descrizione
<code>moduloOstinato(starti, patlen, ltick, tipoostinato, midichan)</code>	ostina la prima nota di ogni gruppo
<code>moduloArp(starti, patlen, ltick, tipoarp, segnoflag, midichan)</code>	arpeggiatore
<code>moduloTetraScala(ltick, ottava, midichan)</code>	scala 0-2-4-6 in quattro tempi

Altri effetti – `moduloArpGlass`, `moduloMedia`, `moduloStrongBeat`, `moduloStrongPattern`, `moduloChangeLast`, `moduloPlayFirstLast`, `moduloNotaperPattern`, `multiAcc` – sono documentati nei commenti in testa a ciascuna funzione in [LibFX.ino](#).

11.4 Moduli liberi

Questi moduli non leggono un pattern ma generano le note direttamente:

`moduloTriangular`, `moduloSinPattern`, `moduloSinfree`, `moduloClusterKeyFull`, `moduloScalaRndKey`, `moduloMahler`, `moduloCluster`, `moduloClusterKey`, `moduloFullRandomNote`, `moduloKeyRandomNote`.

NB Come in tutti i moduli, le note sono messe in coda con `playAdd(nota, midichan, volume, durata, inizio)` e suonano quando arriva il loro tick.

12 Il compositore automatico (complexp)

Oltre ai brani scritti a mano, il sistema contiene un compositore automatico che costruisce un intero brano a partire da pochi parametri “di carattere”: quanto deve essere aggressivo, complesso, moderno o cupo.

12.1 La struttura complexp

La struttura **complexp** è definita in **H_Ard_NG50_Base.ino**; i valori di partenza sono impostati in **setDefaultSong()** (**Funzioni.ino**).

Campo	Descrizione	Default
len	moltiplicatore della lunghezza (1...); lavorare in blocchi multipli	2
aggressivita	0..10: probabilità delle varianti; 0 mai, 10 sempre	5
comp_orizzontale[0]	1..10: complessità dei parametri nelle funzioni	5
comp_verticale[0]	numero di tracce attive 0..6; 10 = casuale	10
modernita	0 tracce classiche, 1 moderne, 2 casuale per sezione, 3 casuale per sezione (0,1,2), 4 casuale per tutto il pezzo (solo 0,1)	3
effetti[0]	0 spento, 1 sempre l'effetto 1, 2 sempre il 2, 3 entrambi, 4 casuale	1
accordi	gruppi attivi 1..3	1
oscurita[0]	0..9: incupisce il suono, agisce su ottava e lunghezza	5

NB Gli elementi [1] sono i valori elaborati/mappati dal sistema: non vanno scritti a mano.

12.2 Effetto sul brano

setDefaultSong() ricava dai parametri le caratteristiche generali del brano:

- **BPM** = $20 + \text{casuale}(0..4) \times 5 + (\text{oscurita mappata}) \times 5$
- **Tonalità** scelta in base all'aggressività: fino a 3 tra le più diffuse, fino a 6 anche i modi, oltre qualunque.

12.3 Sezioni e tracce

Il compositore completo è **modeTrak(base1, sezione, lenpat, pos)** in **SongFunc.ino**; **genMainPatSong()** sceglie la struttura del brano.

Sigla	Sezione	Sigla	Sezione
i	intro	r	ritornello
s	strofa	c	ricorrente
p	ponte	b	break
e	speciale	d	coda
l	solo	f	finale

Traccia	Ruolo
0	accordi
1	melodia

Traccia	Ruolo
2	basso
3-6	accompagnamenti

NB Il compositore completo non è chiamato da `carica()`: attualmente si usano i brani di prova (capitolo 14).

13 Esempio completo di brano

Questo esempio mette insieme quanto visto finora: inizializzazione, tonalità, un pattern scritto a mano con silenzi e un effetto applicato ogni quattro battute.

13.1 Il codice

Aggiungere la funzione, per esempio in un nuovo file **MiaCanzone.ino**, e chiamarla da **carica()** (capitolo 14).

```
void MiaCanzone() {
  byte base1 = 16;                                // tick per battuta
  if (contick == 0) {                              // inizializzazione: una sola volta, al primo tick
    PotAverage[9] = 30;                            // velocità: battute al minuto
                                                    // (1 tick = 60 / (BPM x 16) secondi)
    PotAverage[3] = 0;                             // tonalità: 0 = Do maggiore
    setModeKeySong(PotValue(3));
    DatiVideo.baselen = base1;                    // solo per il display: tick per battuta
    DatiVideo.songlen = 8;                        // solo per il display: battute del brano
    for (int i = 0; i < base1; i++) nota2int(255, i); // parto da un pattern tutto muto...
    nota2int(60, 0); nota2int(60, 1);             // Do Do
    nota2int(62, 4);                               // Re
    nota2int(64, 6); nota2int(64, 7);             // Mi Mi
  }
  if (contick % base1 == 0) {                     // all'inizio di ogni battuta
    moduloBase(0, base1, 1);                      // suona basePat[0..15] sul canale 1
    // la quarta battuta ogni quattro: lo stesso pattern al contrario sul canale 2
    if (tickDivider(4, base1) == 3) moduloBeatRev(0, base1, 0, 0, 2);
  }
}
```

13.2 Come funziona

Parte	Spiegazione
contick == 0	imposta BPM 30 e Do maggiore, svuota il pattern riempiendolo di silenzi e scrive le cinque note
contick % base1 == 0	a ogni battuta (ogni 16 tick) mette in coda il pattern sul canale 1
tickDivider(4, base1) == 3	una battuta ogni quattro aggiunge sul canale 2 lo stesso pattern al contrario

13.3 Risultato

A 30 BPM la battuta dura 2 secondi:

Nota	Inizio	Fine
Do	0 s	0,25 s
Re	0,5 s	0,625 s
Mi	0,75 s	1 s
–	1 s	silenzio fino alla battuta successiva

Sostituendo **moduloBase(0, base1, 1)** con **moduloBase(0, base1, 1, 1, 3)** ogni nota diventa una triade maggiore sul canale 3.

13.4 Ricette rapide

```
// cambiare tonalità a metà brano (Re minore)
PotAverage[3] = 14; setModeKeySong(PotValue(3));
// rallentare
PotAverage[9] = 20;
// accentare i tempi forti
moduloStrongBeat(starti, patlen, countbeat, ltick, midichan);
```

14 Scegliere il brano attivo

In `CaricaF.ino` la funzione `carica()` chiama un solo brano; le chiamate agli altri sono commentate. Per attivare un brano si toglie il `//` dalla sua riga e lo si aggiunge a quella del brano attivo.

Riga in <code>carica()</code>	Descrizione
<code>//PrimoNewMaggiore();</code>	pattern in tonalità maggiore, cambia tono ogni 14 battute
<code>//PrimoNewMinore();</code>	come sopra, in minore
<code>//SingMajMin();</code>	note singole con accordo maggiore/minore, cambia tono
<code>//PlayAgain();</code>	quattro battute con accordi e variazioni, poi cambia tonalità
<code>//Song1();</code>	brano con struttura (intro, strofa, ritornello, ponte); usa <code>diffstop</code> , quindi con le due righe di stop di <code>carica()</code> si ferma al primo tick (vedi 7.5)
<code>MinTest();</code>	brano attivo: pattern generato, espanso e ripetuto, base 32, BPM 30

SUGGERIMENTO Per un brano nuovo: scrivere la funzione (capitolo 13) e chiamarla al posto di `MinTest()`.

15 Cose da sapere

- I comportamenti descritti sono stati verificati in simulazione su PC (tick, note, silenzi, accordi, tasti all'avvio, avvio automatico). Display, rete, sensori e tempi reali non sono stati provati.
- Le pause non occupano posto in **playList**: la loro presenza non rallenta né riempie la lista.
- **BASEPATELE**, **PLAYELE** ed **ENDELE** incidono sulla RAM: sul Mega sono 8 KB in tutto.
- Le funzioni segnalate “NON FUNZIONA” nei commenti originali non sono state modificate.

Appendice A Risoluzione dei problemi

Sintomo	Causa probabile	Rimedio
Non suona nulla, ma la seriale mostra "Cmd: 144 ..."	modalità debug attiva: il MIDI esce come testo	<code>debug = false</code> (4.1)
Il brano non parte all'accensione	<code>acceso = false</code> oppure KEYS_NONE senza comando dal nucleo	<code>acceso = true</code> o premere start (1.1)
Il tasto premuto all'accensione non ha effetto	nei primi 500 ms i tasti sono ignorati	premere dopo l'avvio (1.4)
Le note vanno fuori tempo	il tick dura più dell'intervallo disponibile	controllare <code>Tick max us</code> , escludere display/rete, abbassare il BPM (4.3)
Note mancanti, compare "0"	<code>playlist</code> piena	aumentare <code>PLAYELE</code> sopra il picco (3.4)
Note troppo corte	<code>endList</code> piena: le note vengono spente subito	aumentare <code>ENDELE</code> (3.3)
Song1 si ferma subito	usa <code>diffstop</code> con le righe di stop di <code>carica()</code>	vedi 7.5 e 14
Note uguali di tracce diverse si fondono	stesso <code>midichan</code>	usare canali diversi (7.3)

Appendice B Glossario

Termine	Definizione
tick	unità minima di tempo; dura 60 / (BPM × 16) secondi
base1	numero di tick in una battuta del brano
contick	contatore dei tick dall'inizio del brano
pattern	sequenza di gradi e ottave in basePat , una casella per tick
grado	posizione di una nota nella scala corrente; rende i pattern trasportabili
silenzio	casella con intervallo 255 e ottava 99
esecutore	funzione che legge un pattern e mette in coda le note
effetto	esecutore che trasforma il pattern mentre lo legge
playList	lista delle note in attesa di essere suonate
endList	lista delle note in corso, da spegnere
midichan / tag	canale MIDI e identificativo della traccia
fine differita	arresto del brano al tick successivo tramite diffstop
nucleo	unità esterna collegata su Serial1 che invia comandi di testo
complex	parametri “di carattere” del compositore automatico