MEGA

Project number: IST-1999-20410

GEM EyesWeb WK4 Tools

software contributions to the MEGA SE

GEMWK4 Library, v. 1.0

http://www.generalmusic.com

Contact:
Carlo Drioli

drioli@dei.unipd.it

Introduction

The library WK4Tools is intended to provide software tools for performing and experimenting with the Generalmusic WK4 on-board Vocal Processor. The blocks provided by the GEMWK4 Library are contained in the Sound.GEMWK4 folder in the EyesWeb block browser. The library is based on the System Exclusive protocol reported in Appendix A.

1. The WK4SysEx EyesWeb block

The **WK4SysEx** block provides for the communication between the EyesWeb platform and the WK4 internal Vocal Processor. It implements the communication protocol of WK4, thus allowing for instantaneous amplitude and pitch control for each one of the three additional voices. It has three pairs of control inputs (pitch and amplitude for each voice). The EW implementation is based on the C++ class reported in Appendix B.

2. MIDI control of the WK4 vocal processor

The patch in figure 1.1 can be used to control the Vocal Processor via MIDI. It realizes pitch and amplitude instantaneous control of the VP's additonal voice 1, through the pitch and amplitude control curves provided as inputs. In the case shown, the pitch and amplitude control curves are generated via MIDI by the **MIDI_Adsr** blocks provided in the SoundSynthesis Library. The control of pitch is enhanced by random fluctuations and vibrato generators.

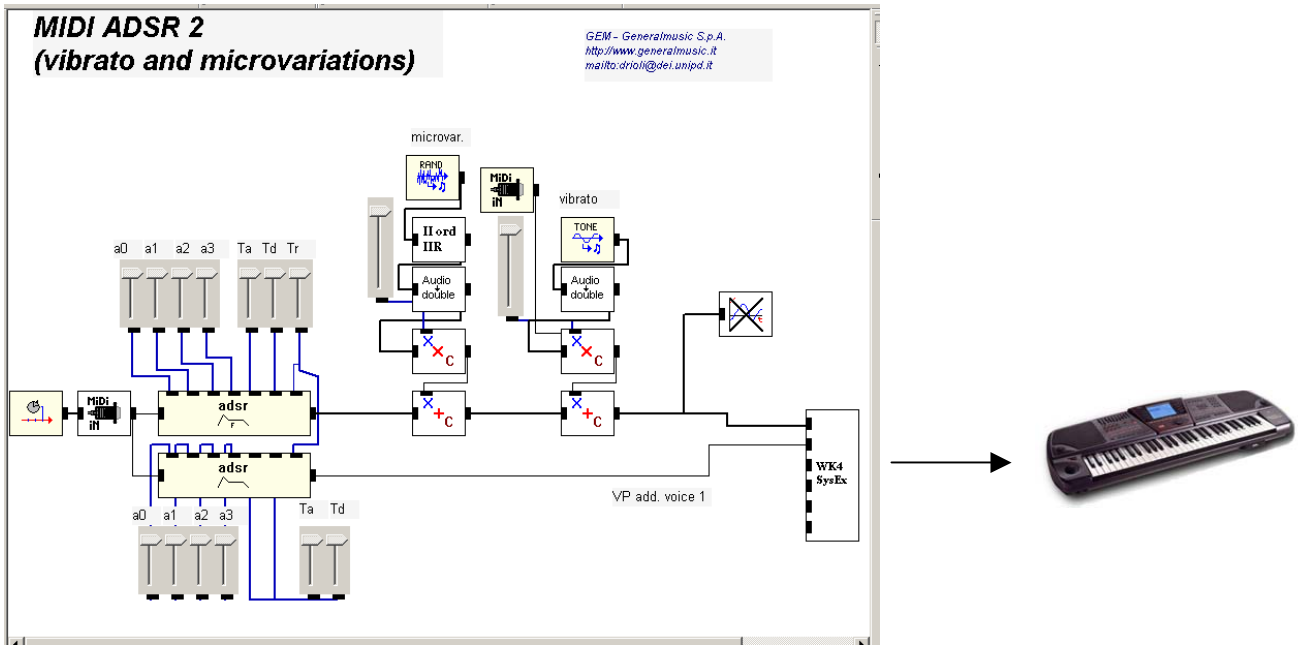

Fig. 1.1: The patch used to ptovide pitch and amplitude real time MIDI control on the WK4 Vocal Processor.

Appendix A: communication protocol

VOCODER SYSTEM EXCLUSIVE MESSAGE TO MIDI IN-A

F0H = System Exclusive Message status
2FH = ID number (manufacturer ID) = GENERAL MUSIC
28H = Length of Data = 40

0000LGDA
0bbbbbbb
0ccccccc **0**Abbbbbbbccccccc **= Voice1Freq**
0eeeeeee
0f f f f f f f **0**Deeeeeeef f f f f f **= Voice1FreqRate**
0hhhhhhh
0 i i i i i i i **0**Ghhhhhhhhi i i i i i i **= Voice1Amp**
0mmmmmmm
0nnnnnnn **0**Lmmmmmmmnnnnnnn **= Voice1AmpRate**
0ppppppp 0ppppppp **= PAN**

0000LGDA
0bbbbbbb
0ccccccc **0**Abbbbbbbccccccc **= Voice2Freq**
0eeeeeee
0f f f f f f f **0**Deeeeeeef f f f f f **= Voice2FreqRate**
0hhhhhhh
0 i i i i i i i **0**Ghhhhhhhhi i i i i i i **= Voice2Amp**
0mmmmmmm
0nnnnnnn **0**Lmmmmmmmnnnnnnn **= Voice2AmpRate**
0ppppppp 0ppppppp **= PAN**

0000LGDA
0bbbbbbb
0ccccccc **0**Abbbbbbbccccccc **= Voice3Freq**
0eeeeeee
0f f f f f f f **0**Deeeeeeef f f f f f **= Voice3FreqRate**
0hhhhhhh
0 i i i i i i i **0**Ghhhhhhhhi i i i i i i **= Voice3Amp**
0mmmmmmm
0nnnnnnn **0**Lmmmmmmmnnnnnnn **= Voice3AmpRate**
0ppppppp 0ppppppp **= PAN**

0000LGDA
0bbbbbbb
0ccccccc **0**Abbbbbbbccccccc **= Voice4Freq**
0eeeeeee
0f f f f f f f **0**Deeeeeeef f f f f f **= Voice4FreqRate**
0hhhhhhh
0 i i i i i i i **0**Ghhhhhhhhi i i i i i i **= Voice4Amp**
0mmmmmmm
0nnnnnnn **0**Lmmmmmmmnnnnnnn **= Voice4AmpRate**
0ppppppp 0ppppppp **= PAN**

F7H = EOX (End Of Exclusive)

Spedizione nota cantata su midi OUT-B

NoteOn Chn=16 -- Notecode=xx -- Dynamic=64
9F xx 40

THE VOCODER PARAMETER STRUCTURE

```
struct TO_DSP_NOTE_PAR
{
 WORD  Voice1Freq;          /* frequency  */
 WORD  Voice1FreqRate;      /* rate della rampa per il parametro di freq */
 WORD  Voice1Amp;           /* amplitude */
 WORD  Voice1AmpRate;       /* rate della rampa per il parametro di amp */
 BYTE  Pan;                 7* 0…..0x40…..7F  */

 WORD  Voice2Freq;          /* frequency  */
 WORD  Voice2FreqRate;      /* rate della rampa per il parametro di freq */
 WORD  Voice2Amp;           /* amplitude */
 WORD  Voice2AmpRate;       /* rate della rampa per il parametro di amp */
 BYTE  Pan;                 7* 0…..0x40…..7F  */

 WORD  Voice3Freq;          /* frequency  */
 WORD  Voice3FreqRate;      /* rate della rampa per il parametro di freq */
 WORD  Voice3Amp;           /* amplitude */
 WORD  Voice3AmpRate;       /* rate della rampa per il parametro di amp */
 BYTE  Pan;                 7* 0…..0x40…..7F  */

 WORD  Voice4Freq;          /* frequency  */
 WORD  Voice4FreqRate;      /* rate della rampa per il parametro di freq */
 WORD  Voice4Amp;           /* amplitude */
 WORD  Voice4AmpRate;       /* rate della rampa per il parametro di amp */
 BYTE  Pan;                 7* 0…..0x40…..7F  */
};
```

1) **WORD** = 16bit -> 15utili **BYTE =** 8bit -> 7utili

2) Voice**x**Freq Ogni unità exprime 1/64 di semitono. Vedi file PITCHTAB.C

3) Voice**x**FreqRate In WK8 con LFO spento viene usato il valore massimo 0x7FFF, con LFO acceso invece viene usato il valore 0x333.

4) Voice**x**Amp Per avere una risposta corretta questo valore deve essere scalato in **dB**. Mettendo a 0 questo valore la nota viene spenta. In WK8 per valori di ampiezza inferiori a 0x20 l' LFO non entrava.

5) Voice**x**AmpRate In WK8 con LFO spento viene usato il valore 0x1000, con LFO acceso viene usato il valore 0x333, quando si spegne una nota viene usato il valore 0x100.

6) Pan Range = 0…..0x40…..0x7F

- A fronte dei circa 15ms necessari per la trasmissione midi dei 44 Byte del pacchetto Exclusivo la WK8 rinfresca i dati al DSP ogni 2ms.

Appendix B: C++ class implementing the communication protocol

```
//
// Copyright (c) Generalmusic S.p.A.
//
// Author: Carlo Drioli
// Date: 4/6/2001
// http://www.generalmusic.it
// e-mail:  drioli@dei.unipd.it
//
//    GemWK4tools.h
//

#if !defined __GEMVP_TOOLS_INCLUDED__
#define __GEMVP_TOOLS_INCLUDED__


typedef struct VP_Message
{
      float   VoiceFreq[4];      /* frequency  */
      float   VoiceFreqRate[4];  /* rate della rampa per il parametro di freq */
      float   VoiceAmp[4];       /* amplitude */
      float   VoiceAmpRate[4];   /* rate della rampa per il parametro di amp */
      char  Pan[4];              /* 0…..0x40…..7F  */
} VP_Message;

class CVPMessage
{
      public:
            CVPMessage();
            ~CVPMessage();
            void MessageInit();
            void MessageClear();
            void AmpFreq2WK4SysExMess(const int voice, const double Amp, const
double Freq);
            void SetAmp(const int voice, const float Amp);
            unsigned short SetFreq(const int voice, const float Freq);
            char* GiveSysExMessage();
      private:
            VP_Message m_VPMess;
            char m_VPSysExMess[44];
};

// tabella di 64 valori per ogni semitono
const unsigned short PITCH_TAB[] = {
 // Midi Code  24 (C1, 32.7032 Hz)
 512, ... ... 65477,
};
#endif //!defined
```

```
//
// Copyright (c) Generalmusic S.p.A.
//
// Author: Carlo Drioli
// Date: 4/6/2001
// http://www.generalmusic.it
// e-mail:  drioli@dei.unipd.it
//
//    GemWK4tools.cpp
//

#include "GemWK4tools.h"
#include <iostream.h>
#include <math.h>

//////////////////////////////////////////////////////////////////////
// Class definition
//////////////////////////////////////////////////////////////////////
CVPMessage::CVPMessage()
{
      MessageInit();
}

CVPMessage::~CVPMessage()
{
}

void CVPMessage::MessageInit()
{
      int i;
      for (i=0;i<4;i++)
      {
            m_VPMess.VoiceFreq[i]=0;
            m_VPMess.VoiceFreqRate[i]=0;
            m_VPMess.VoiceAmp[i]=0;
            m_VPMess.VoiceAmpRate[i]=0;
            m_VPMess.Pan[i]=64;
      }

      m_VPSysExMess[0]=(char)0xf0;
      m_VPSysExMess[1]=(char)0x2f;
      m_VPSysExMess[2]=(char)0x28;
      for(i=3; i<43; i++)
            m_VPSysExMess[i]=(char)0x0;

      m_VPSysExMess[12]=(char)0x40; //Pan
      m_VPSysExMess[22]=(char)0x40;
      m_VPSysExMess[32]=(char)0x40;
      m_VPSysExMess[42]=(char)0x40;
      m_VPSysExMess[43]=(char)0xf7;
}

void CVPMessage::MessageClear()
{
      int i;
      for (i=0;i<4;i++)
      {
            m_VPMess.VoiceFreq[i]=0;
            m_VPMess.VoiceFreqRate[i]=0;
            m_VPMess.VoiceAmp[i]=0;
            m_VPMess.VoiceAmpRate[i]=0;
            m_VPMess.Pan[i]=64;
      }
```

```
	for(i=3; i<43; i++)
		m_VPSysExMess[i]=(char)0x0;
	m_VPSysExMess[12]=(char)0x40; //Pan
	m_VPSysExMess[22]=(char)0x40;
	m_VPSysExMess[32]=(char)0x40;
	m_VPSysExMess[42]=(char)0x40;
}

void CVPMessage::SetAmp(const int voice, const float Amp)
{
	m_VPMess.VoiceAmp[voice]=Amp;
	m_VPMess.VoiceAmpRate[voice]=0;

	int sysexindex;
	//Amp
	sysexindex=8+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)(((unsigned short)Amp >> 7) & 0x7f);
	sysexindex=9+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)(((unsigned short)Amp) & 0x7f);

	//AmpRate
	sysexindex=10+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0x10;
	sysexindex=11+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0x00;

	sysexindex=3+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0x00; // TO DO:aggiorna LGDA
}

unsigned short CVPMessage::SetFreq(const int voice, const float Freq)
{
	m_VPMess.VoiceFreq[voice]=Freq;
	m_VPMess.VoiceFreqRate[voice]=0;
	double n = (69.0+12.0*log(fabs(Freq)/440.0)/log(2.0));
	double remn = n-(int)n;

	int pitchtableindex=((int)n-24)*64+(int)(remn*64.0);
	if (pitchtableindex<0)
		pitchtableindex=0;

	unsigned short VPNoteCode=PITCH_TAB[pitchtableindex];
	int sysexindex;
	//Freq
	sysexindex=4+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)((VPNoteCode >> 7) & 0x7f);
	sysexindex=5+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)(VPNoteCode & 0x7f);
	//FreqRate
	sysexindex=6+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0x03;
	sysexindex=7+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0xff;
	sysexindex=3+(10*(voice-1));
	m_VPSysExMess[sysexindex]=(char)0x00; //TO DO: aggiorna LGDA
	return VPNoteCode;
}

char* CVPMessage::GiveSysExMessage()
{
return m_VPSysExMess;
}
```