***Auto.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "ScriptPhase.h"

#include "Delay.h"

#include "Composer.h"

#include "SpecialKeyScan.h"

#include "KeyDisp.h"

#include "VolDisp.h"

#include "Menu.h"

#include "SpecialKeyScan.h"

#include "WaveSel.h"

#include "LCD.h"

const char Carry[]={"63+=.5+==3+=6=1+6==1+==6==1+==3+3+==2+==3+==1+==63+.=5+==6=+6=+6=+3=+5+3+==5+==3+==5+==6+=6+=6+=3+=5+3+x5+=6=3+x5+=6=3+=2#+==3+==1+2+2+2+=1+==2+==3+=5+=6+3++3+3++3+=3++=3+=3++=3+=2+==#3+==1+2+2+2+=1+==2+==3+=5+=6+~"};

const char Farewell[]={"53=5=1+~61+5~51=2=32=1=2~0053=5=1+.7=61+5~52=3=4.7=-1~0061+1+~76=7=1+~6=7=1+=6=6=5=3=1=2~0053=5=1+.7=61+5~52=3=4.7=-1~~0"};

const char Represent[]={"0=5-=5-x1.3=3x5.1=7-.3=50=5=5x6=71+.6=6=6x5=5~~3=2=1.1=13=2=1.1=12=2x3=3x21=6-2=3=3x2~~0=5-=5x1.3=3x5.1=7-.3=50=5=5x6=71+.6=6=6x5=5~~3=2=1.1=13=2=1.1=12=2x3=3x2.6=-7-1=2=2x1~~3=5=3.2=157-~~6=-7=-6-.7=-6-.5-3~~53.2=157-~~6=-7=-1.1=12=3=2~~0=5-=5-x1.3=3x5.1=7.-3=5.5=5x6.7=1+.6=6=6x5=5~3=2=1.1=13=2=41.1=12=3=3x2.6=-7-1=2=1~~0"};

const char BlueGeniue[]={"3==4==5=3+=1+=5=3.2==3==4=2+=7=4=2+.1==2==3=2==3==4=3==4==5=3==4==6=5==6==7==1+==1+#==2+==2+==3+==4+==4+#==5+=0=5-1=3=5-=3=1=3=5-=3==4==5=4=3=4=5=4=3=4=5=5==4#==5=3=7.2==3==4=3=5=2+=7.2==3==4=3=4=6=5.3==4==5=4=3=4=5=4=3=4=5=5==4#==5=3+=2+.2==3==4=3=4=2+=1+=7=6=7=1+~1+553+=3+=1+53~"};

const char BadApple[]={"6=7=1+=2+=3+6+=5+=3+6=.0==3+=2+=1+=7=6=7=1+=2+=3+.=0==2+=1+=7=6=7=1+=7=6=5#=7=6=7=1=+2+=3+=.0==6+=5+=3+.=0==63+=2+=1+=7=6=7=1+=2+=3+=.0==2+=1+=7=.0==1+=.0==2+3+6=7=1+=2=+3+6+=5+=3+6=.0==3+=2+=1+=7=6=7=1+=2+=3+=.0==2+=1+=7=6=7=1+=7=6=5#=7=6=7=1+=2+=3+=.0==6+=5+=3+=.0==63+=2+=1+=7=6=7=1+=2+=3+=.0==2+=1+=7=.0==1+.=0==2+3+1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==2++==7+==2++==7+==2++==7+==2++==7+==2++==7+==2++==7+==2++==7+==2++==7+==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==3++==1++==2++=1++==2++==1++=6+==7+==6+==4+==6+==4+==6+==4+==6+==4+==6+==4+==6+==4+==6+==4+==6+==4+==7+==5+==7+==5+==7+==5+==7+==5+==7+==5+==7+==5+==7+==5+==7+==5+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==1++==6+==6+==1+==6+==0==6+=3+==1+==6==0==0==3+==5+=6+=3+=2+=3+2+=3+=5+=6+=3+=2+=3+.=0==2+=3+=2+=1+=7=6=6.=0==5=6=7=1+=2+=3+=6.=0==2+=3+=5+=6+=3+=2+=3+2+=3+=5+=6+=3+=2+=3+6+=7+=1++=7+=6+=5+=3+.=0==2+=3+=2+=1+=7=5=6.0=6+~~~5+~~2+2+=3+.3+~0=0=2+=1+=7=5=2=3=6~~6=7==1.==2+~~2+==1+==7==5==6~~00==7.=1+=2+=3+=6+.6.~~6+=5+==4+==3+~2+1+.=0==7.=1+==1+=5+=5+~3+~~2+=3+=5+=6+=3+=2+=3+.=0==2+=3+=5+=6+=3+=2+=3+.=0==6+=7+=1++=7+=6+=5+=3+.=0==2+=3+=2+=1+=7=5=60"};

const char Test[]={"11#22#344#55#66#71+1+#2+2+#3+4+4+#5+5+#6+6+#7+1++1++#2++2++#3++4++4++#5++5++#6++6++#7++"};

void MenuCallbackPlay(u32 Tag);

const struct MenuStruct SongList[]={

"Pigman Carry Wife",&MenuCallbackPlay,(u32)Carry,

"Farewell",&MenuCallbackPlay,(u32)Farewell,

"Represent",&MenuCallbackPlay,(u32)Represent,

"BlueGeniue",&MenuCallbackPlay,(u32)BlueGeniue,

"BadApple",&MenuCallbackPlay,(u32)BadApple,

"Test",&MenuCallbackPlay,(u32)Test,

0,0,0,

};

void Play(const char\* Song)

{

RefreshKeyDisp();

VolDisp(GetVol());

LCDDrawText(10,10,GREEN,GRAY2,"Auto Play:");

LCDDrawText(10,30,GREEN,GRAY2,"Keyboard disabled.");

LCDDrawText(10,50,GREEN,GRAY2,"Left=Back");

LCDDrawText(10,70,GREEN,GRAY2,"Right=Change wave type");

LoadScript(Song);

PreciseDelay(0);

while(1)

{

u8 Ctrl=SpecialKeyGetCtrl();

signed short Enc=SpecialKeyGetEnc();

if(Ctrl & 1)

{

break;

}

if(Ctrl & 2)

{

SetAllTuneDisable();

SetAllKeyDispDisable();

WaveSel();

RefreshKeyDisp();

VolDisp(GetVol());

LCDDrawText(10,10,GREEN,GRAY2,"Auto Play:");

LCDDrawText(10,30,GREEN,GRAY2,"Keyboard disabled.");

LCDDrawText(10,50,GREEN,GRAY2,"Left=Back");

LCDDrawText(10,70,GREEN,GRAY2,"Right=Change wave type");

}

if(Enc)

{

signed short Vol=GetVol()+Enc\*5;

if(Vol>255)

{

Vol=255;

}

if(Vol<0)

{

Vol=0;

}

SetVol(Vol);

VolDisp(Vol);

}

if(ScriptTick()!=1)

{

LoadScript(Song);

Delay(1000);

}

PreciseDelay(10);

}

SetAllTuneDisable();

SetAllKeyDispDisable();

}

void MenuCallbackPlay(u32 Tag)

{

Play((const char\*)Tag);

}

void DispSongList(void)

{

EnterMenu(SongList);

}

***Composer.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "stm32f10x\_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO

#include "stm32f10x\_tim.h" // Keil::Device:StdPeriph Drivers:TIM

#include "stm32f10x\_dac.h" // Keil::Device:StdPeriph Drivers:DAC

#include "stm32f10x\_dma.h" // Keil::Device:StdPeriph Drivers:DMA

#include <math.h>

#include <string.h>

#define PI 3.14159265358979323846

#define DHR12R1\_OFFSET ((uint32\_t)0x00000008)

#define DACMax ((uint32\_t)0x00000FFF)

#define Adjust (-18)

#define MaxActive 5

#define Scale 1000000

#define FrepToStep(Freq) (Scale\*(Freq)/500)

#define Step 200

#define TuneCount 40

#define BufferSize 0x500

const u32 TuneStep[TuneCount]={

FrepToStep(262),FrepToStep(277),FrepToStep(294),FrepToStep(311),FrepToStep(330),FrepToStep(349),FrepToStep(370),FrepToStep(392),FrepToStep(415),FrepToStep(440),FrepToStep(466),FrepToStep(494),

FrepToStep(523),FrepToStep(554),FrepToStep(578),FrepToStep(622),FrepToStep(659),FrepToStep(698),FrepToStep(740),FrepToStep(784),FrepToStep(831),FrepToStep(880),FrepToStep(932),FrepToStep(988),

FrepToStep(1047),FrepToStep(1109),FrepToStep(1175),FrepToStep(1245),FrepToStep(1319),FrepToStep(1397),FrepToStep(1480),FrepToStep(1568),FrepToStep(1661),FrepToStep(1760),FrepToStep(1865),FrepToStep(1976),

//Only for test porpose

FrepToStep(250),FrepToStep(500),FrepToStep(1000),FrepToStep(2000),

};

signed int WaveTable[Step];

u16 Buffer[BufferSize];

u8 Tunes[TuneCount];

u32 TuneCounter[TuneCount];

u8 ActiveCount;

u16 Volume;

u8 Vol8;

u8 WaveType;

void InitGPIO(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);

GPIO\_InitStruct.GPIO\_Mode=GPIO\_Mode\_AIN;

GPIO\_InitStruct.GPIO\_Pin=GPIO\_Pin\_4;

GPIO\_InitStruct.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOA,&GPIO\_InitStruct);

}

void InitDAC(void)

{

DAC\_InitTypeDef DAC\_InitStruct;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_DAC, ENABLE);

DAC\_InitStruct.DAC\_LFSRUnmask\_TriangleAmplitude=0;

DAC\_InitStruct.DAC\_OutputBuffer=DAC\_OutputBuffer\_Enable;

DAC\_InitStruct.DAC\_Trigger=DAC\_Trigger\_T6\_TRGO;

DAC\_InitStruct.DAC\_WaveGeneration=DAC\_WaveGeneration\_None;

DAC\_Init(DAC\_Channel\_1,&DAC\_InitStruct);

}

void InitNVIC(void)

{

NVIC\_InitTypeDef NVIC\_InitStructure;

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_0);

NVIC\_InitStructure.NVIC\_IRQChannel=DMA2\_Channel3\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority=0;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority=0;

NVIC\_InitStructure.NVIC\_IRQChannelCmd=ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

}

void InitDMA(void)

{

DMA\_InitTypeDef DMA\_InitStruct;

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_DMA2, ENABLE);

DMA\_InitStruct.DMA\_BufferSize=BufferSize;

DMA\_InitStruct.DMA\_DIR=DMA\_DIR\_PeripheralDST;

DMA\_InitStruct.DMA\_M2M=DMA\_M2M\_Disable;

DMA\_InitStruct.DMA\_MemoryBaseAddr=(uint32\_t)Buffer;

DMA\_InitStruct.DMA\_MemoryDataSize=DMA\_MemoryDataSize\_HalfWord;

DMA\_InitStruct.DMA\_MemoryInc=DMA\_MemoryInc\_Enable;

DMA\_InitStruct.DMA\_Mode=DMA\_Mode\_Circular;

DMA\_InitStruct.DMA\_PeripheralBaseAddr=(uint32\_t)DAC\_BASE+DHR12R1\_OFFSET+DAC\_Align\_12b\_R;

DMA\_InitStruct.DMA\_PeripheralDataSize=DMA\_PeripheralDataSize\_HalfWord;

DMA\_InitStruct.DMA\_PeripheralInc=DMA\_PeripheralInc\_Disable;

DMA\_InitStruct.DMA\_Priority=DMA\_Priority\_VeryHigh;

DMA\_Init(DMA2\_Channel3,&DMA\_InitStruct);

DMA\_ITConfig(DMA2\_Channel3,DMA\_IT\_TC | DMA\_IT\_HT,ENABLE);

}

void InitTimer(void)

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseInitStruct;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM6, ENABLE);

TIM\_TimeBaseInitStruct.TIM\_ClockDivision=TIM\_CKD\_DIV1;

TIM\_TimeBaseInitStruct.TIM\_CounterMode=TIM\_CounterMode\_Up;

TIM\_TimeBaseInitStruct.TIM\_Period=12000000/100000+Adjust;//100KHz

TIM\_TimeBaseInitStruct.TIM\_Prescaler=6;

TIM\_TimeBaseInitStruct.TIM\_RepetitionCounter=0x00;

TIM\_TimeBaseInit(TIM6,&TIM\_TimeBaseInitStruct);

TIM\_SelectOutputTrigger(TIM6,TIM\_TRGOSource\_Update);

}

void InitWaveTableSine(void)

{

u32 i=0;

for(;i<Step;i++)

{

WaveTable[i]=sin(2\*PI\*i/Step)\*Volume;

}

}

void InitWaveTableSquare(void)

{

u32 i=0;

for(;i<Step;i++)

{

WaveTable[i]=i<Step/2?Volume:-(signed int)Volume;

}

}

void InitWaveTableTriangle(void)

{

u32 i=0;

for(;i<Step;i++)

{

if(i<Step/4)

{

WaveTable[i]=4\*i\*Volume/Step;

}

else if(i<3\*Step/4)

{

WaveTable[i]=Volume-4\*(i-Step/4)\*Volume/Step;

}

else

{

WaveTable[i]=-(signed int)(4\*(Step-i)\*Volume/Step);

}

}

}

void InitWaveTableSawTooth(void)

{

u32 i=0;

for(;i<Step;i++)

{

WaveTable[i]=i\*2\*Volume/Step-Volume;

}

}

void InitWaveTableDualSine(void)

{

u32 i=0;

for(;i<Step;i++)

{

WaveTable[i]=sin(2\*PI\*i/Step)\*Volume\*2/3+sin(4\*PI\*i/Step)\*Volume/3;

}

}

void RefreshWaveTable(void)

{

switch(WaveType)

{

default:

case 0:

InitWaveTableSine();

break;

case 1:

InitWaveTableSquare();

break;

case 2:

InitWaveTableTriangle();

break;

case 3:

InitWaveTableSawTooth();

break;

case 4:

InitWaveTableDualSine();

break;

}

}

void SetWaveType(u8 Type)

{

WaveType=Type;

RefreshWaveTable();

}

void InitComposer(void)

{

ActiveCount=0;

memset(Buffer,0,sizeof(Buffer));

memset(Tunes,0xFF,sizeof(Tunes));

memset(TuneCounter,0,sizeof(TuneCounter));

InitGPIO();

InitDAC();

InitNVIC();

InitDMA();

InitTimer();

DMA\_Cmd(DMA2\_Channel3,ENABLE);

DAC\_Cmd(DAC\_Channel\_1,ENABLE);

DAC\_DMACmd(DAC\_Channel\_1,ENABLE);

TIM\_Cmd(TIM6,ENABLE);

Volume=0;

Vol8=0;

WaveType=0;

RefreshWaveTable();

}

void ConmosePart(u8 FirstPart)

{

u16\* Base=FirstPart?Buffer:Buffer+BufferSize/2;

u16 i=0;

for(;i<BufferSize/2;i++)

{

u8 j=0;

signed int CurVal=0;

for(;j<ActiveCount;j++)

{

u8 k=Tunes[j];

TuneCounter[k]+=TuneStep[k];

if(TuneCounter[k]>=Step\*Scale)

{

TuneCounter[k]-=Step\*Scale;

}

CurVal+=WaveTable[TuneCounter[k]/Scale];

}

CurVal/=ActiveCount;

Base[i]=DACMax/2+CurVal;

}

}

void Compose(void)

{

ConmosePart(1);

ConmosePart(0);

}

u8 SetTuneEnable(u8 Index,u8 Enable)

{

u8 Success=0;

if(Index<TuneCount)

{

u8 i=0;

if(Enable)

{

if(ActiveCount>MaxActive)

{

return 0;

}

Success=1;

for(;i<ActiveCount;i++)

{

if(Tunes[i]==Index)

{

Success=0;

break;

}

}

if(Success)

{

Tunes[i]=Index;

ActiveCount++;

}

}

else

{

for(;i<ActiveCount;i++)

{

if(Tunes[i]==Index)

{

Success=1;

break;

}

}

if(Success)

{

for(;i<ActiveCount-1;i++)

{

Tunes[i]=Tunes[i+1];

}

Tunes[ActiveCount-1]=0xFF;

ActiveCount--;

}

}

}

return 1;

}

void SetAllTuneDisable(void)

{

ActiveCount=0;

}

void SetVol(u8 Vol)

{

if(Vol!=Vol8)

{

Vol8=Vol;

Volume=DACMax\*Vol/2/255-80;

if(Volume>DACMax)

{

Volume=0;

}

RefreshWaveTable();

}

}

u8 GetVol(void)

{

return Vol8;

}

***Delay.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "SysTick.h"

void Delay(u32 nTime)

{

u32 EndTicks=GetSysTick()+nTime\*10;

while(GetSysTick()<EndTicks);

}

void PreciseDelay(u32 nTime)

{

static u32 EndTicks=0;

while(GetSysTick()<EndTicks);

EndTicks=GetSysTick()+nTime\*10;

}

void Delay100us(u32 nTime)

{

u32 EndTicks=GetSysTick()+nTime;

while(GetSysTick()<EndTicks);

}

***KeyDisp.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "LCD.h"

#include <string.h>

#define FullTuneTextColor BLACK

#define FullTuneColorNormal WHITE

#define FullTuneColorPressed RED

#define HalfTuneTextColor (BLUE | GREEN)

#define HalfTuneColorNormal BLACK

#define HalfTuneColorPressed 0x8000

#define TuneCount 36

#define FullTuneCount 21

#define HalfTuneCount 15

#define FullKeyHeight 100

#define HalfKeyHeight 70

const u8 StartY=Y\_MAX\_PIXEL-FullKeyHeight-3;

const u8 KeyWidth=X\_MAX\_PIXEL/FullTuneCount;

const u8 Marge=(X\_MAX\_PIXEL-(KeyWidth\*FullTuneCount))/2;

const u8 KeyProp[TuneCount]={

1, 0, 2, 0, 3, 4, 0, 5, 0, 6, 0, 7,

8, 0, 9, 0,10,11, 0,12, 0,13, 0,14,

15, 0,16, 0,17,18, 0,19, 0,20, 0,21,

};

const u32 Decoration[]={0x562B,0x73AE,0xDEA3,0x1028,0xF7FB,0x4108,0xDDAB,0x492E,0x7F6A,0x892A,0xFDEB,0x0480,0xFDFB,0x048A,0xDEEF,0x50A2};

u8 LastState[TuneCount];

void PrintNumberHint(u16 X,u16 EndY,u8 Index,u16 TextColor,u16 BGColor)

{

char Num[2]={" \0"};

u8 Tune=KeyProp[Index];

if(Tune==0)

{

Index--;

Tune=KeyProp[Index];

LCDDrawText(X+4,EndY-48,TextColor,BGColor,"#");

}

if(Tune>14)

{

LCDDrawText(X+4,EndY-32,TextColor,BGColor,":");

Tune-=14;

}

else if(Tune>7)

{

LCDDrawText(X+4,EndY-32,TextColor,BGColor,".");

Tune-=7;

}

Num[0]=Tune+'0';

LCDDrawText(X+4,EndY-16,TextColor,BGColor,Num);

}

void PrintKey(u8 Index)

{

u16 X,Height;

u16 Color;

if(KeyProp[Index])

{

X=Marge+(KeyProp[Index]-1)\*KeyWidth;

Height=FullKeyHeight;

Color=LastState[Index]?FullTuneColorPressed:FullTuneColorNormal;

}

else

{

X=Marge+(KeyProp[Index-1]-1)\*KeyWidth+KeyWidth/2;

Height=HalfKeyHeight;

Color=LastState[Index]?HalfTuneColorPressed:HalfTuneColorNormal;

}

LCDFill(X+1,StartY+1,X+KeyWidth-1,StartY+Height-1,Color);

if(LastState[Index])

{

PrintNumberHint(X,StartY+Height-2,Index,KeyProp[Index]?FullTuneTextColor:HalfTuneTextColor,Color);

}

}

void SetKeyDisp(u8 Index,u8 Enable)

{

if(Index>TuneCount)

{

return;

}

if(LastState[Index]==Enable)

{

return;

}

else

{

LastState[Index]=Enable;

}

PrintKey(Index);

if(Index>0)

{

if(KeyProp[Index-1]==0)

{

PrintKey(Index-1);

}

}

if(Index<TuneCount-1)

{

if(KeyProp[Index+1]==0)

{

PrintKey(Index+1);

}

}

}

void DrawDecoration(void)

{

u16 X,Y;

for(Y=0;Y<StartY-5;Y++)

{

for(X=0;X<X\_MAX\_PIXEL;X++)

{

if(Decoration[Y/3%16] & 0x8000>>(X/3%16))

{

LCDDrawPoint(X,Y,BLUE | RED);

}

}

}

}

void RefreshKeyDisp(void)

{

u8 i;

LCDClear(GRAY2);

DrawDecoration();

LCDFill(0,StartY-5,X\_MAX\_PIXEL,Y\_MAX\_PIXEL,GRAY1);

for(i=0;i<TuneCount;i++)

{

if(KeyProp[i])

{

PrintKey(i);

}

}

for(i=0;i<TuneCount;i++)

{

if(!KeyProp[i])

{

PrintKey(i);

}

}

}

void SetAllKeyDispDisable(void)

{

memset(LastState,0,sizeof(LastState));

}

***KeyScan.c***

#include "Device/Include/stm32f10x.h" // Device header

#include <string.h>

u8 KeyVals[36];

void KeyScanInit(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO | RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_GPIOB | RCC\_APB2Periph\_GPIOE | RCC\_APB2Periph\_GPIOD, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_All;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IPU;

GPIO\_Init(GPIOB, &GPIO\_InitStructure);

GPIO\_Init(GPIOE, &GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_8 | GPIO\_Pin\_9 | GPIO\_Pin\_10 | GPIO\_Pin\_11 | GPIO\_Pin\_12 | GPIO\_Pin\_15;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0 | GPIO\_Pin\_1 | GPIO\_Pin\_2 | GPIO\_Pin\_3 | GPIO\_Pin\_4 | GPIO\_Pin\_5;

GPIO\_Init(GPIOD, &GPIO\_InitStructure);

GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable,ENABLE);

GPIO\_SetBits(GPIOB, GPIO\_Pin\_All);

GPIO\_SetBits(GPIOE, GPIO\_Pin\_All);

memset(KeyVals,0,sizeof(KeyVals));

}

u8\* KeyScanGetVals(void)

{

u8 i;

u32 Data=GPIO\_ReadInputData(GPIOB) | (u32)GPIO\_ReadInputData(GPIOE)<<16;

for(i=0;i<24;i++)

{

KeyVals[i]=Data & 0x1?0:1;

Data>>=1;

}

Data=GPIO\_ReadInputData(GPIOA)>>8;

for(;i<29;i++)

{

KeyVals[i]=Data & 0x1?0:1;

Data>>=1;

}

Data>>=2;

KeyVals[i]=Data & 0x1?0:1;

i++;

Data=GPIO\_ReadInputData(GPIOD);

for(;i<36;i++)

{

KeyVals[i]=Data & 0x1?0:1;

Data>>=1;

}

return KeyVals;

}

***LCD.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "LCD.h"

#include "Delay.h"

#include "FONT.h"

u16 SPI\_WriteDat(u16 Dat)

{

while (SPI\_I2S\_GetFlagStatus(SPI1, SPI\_I2S\_FLAG\_TXE) == RESET);

SPI\_I2S\_SendData(SPI1, Dat);

while (SPI\_I2S\_GetFlagStatus(SPI1, SPI\_I2S\_FLAG\_RXNE) == RESET);

return SPI\_I2S\_ReceiveData(SPI1);

}

void LcdGPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOD, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = LCDRS|LCDCS|LCDRST|LCDBG;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

GPIO\_Init(LCDCTRL, &GPIO\_InitStructure);

}

void SPIInit(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

SPI\_InitTypeDef SPI\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);

/\* Configure SPI1 pins: SCK and MOSI \*/

GPIO\_InitStructure.GPIO\_Pin = LCDSCK | LCDMOSI;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

/\* Configure SPI pins: MISO \*/

GPIO\_InitStructure.GPIO\_Pin = LCDMISO;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IPU;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

/\* Enable SPI clocks \*/

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_SPI1, ENABLE);

/\* SPI1 configuration \*/

SPI\_InitStructure.SPI\_Direction = SPI\_Direction\_2Lines\_FullDuplex;

SPI\_InitStructure.SPI\_Mode = SPI\_Mode\_Master;

SPI\_InitStructure.SPI\_DataSize = SPI\_DataSize\_8b;

SPI\_InitStructure.SPI\_CPOL = SPI\_CPOL\_High;

SPI\_InitStructure.SPI\_CPHA = SPI\_CPHA\_2Edge;

SPI\_InitStructure.SPI\_NSS = SPI\_NSS\_Soft;

SPI\_InitStructure.SPI\_BaudRatePrescaler = SPI\_BaudRatePrescaler\_2;

SPI\_InitStructure.SPI\_FirstBit = SPI\_FirstBit\_MSB;

SPI\_InitStructure.SPI\_CRCPolynomial = 7;

SPI\_Init(SPI1, &SPI\_InitStructure);

/\* Enable SPI1 \*/

SPI\_Cmd(SPI1, ENABLE);

}

void LCDReset(void)

{

GPIO\_ResetBits(LCDCTRL,LCDRST);

Delay(100);

GPIO\_SetBits(LCDCTRL,LCDRST);

Delay(50);

}

void LCDWriteIndex(u8 Index)

{

GPIO\_ResetBits(LCDCTRL,LCDRS);

GPIO\_ResetBits(LCDCTRL,LCDCS);

SPI\_WriteDat(Index);

GPIO\_SetBits(LCDCTRL,LCDCS);

}

void LCDWriteData(u8 Data)

{

GPIO\_SetBits(LCDCTRL,LCDRS);

GPIO\_ResetBits(LCDCTRL,LCDCS);

SPI\_WriteDat(Data);

GPIO\_SetBits(LCDCTRL,LCDCS);

}

void LCDWriteData16Bit(u16 Data)

{

GPIO\_SetBits(LCDCTRL,LCDRS);

GPIO\_ResetBits(LCDCTRL,LCDCS);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_16b);

SPI\_WriteDat(Data);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_8b);

GPIO\_SetBits(LCDCTRL,LCDCS);

}

void LCDWriteIndex16Bit(u16 Data)

{

GPIO\_ResetBits(LCDCTRL,LCDRS);

GPIO\_ResetBits(LCDCTRL,LCDCS);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_16b);

SPI\_WriteDat(Data);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_8b);

GPIO\_SetBits(LCDCTRL,LCDCS);

}

void LCDInit(void)

{

LcdGPIO\_Init();

SPIInit();

LCDReset();

LCDWriteIndex(0xCB);

LCDWriteData(0x39);

LCDWriteData(0x2C);

LCDWriteData(0x00);

LCDWriteData(0x34);

LCDWriteData(0x02);

LCDWriteIndex(0xCF);

LCDWriteData(0x00);

LCDWriteData(0XC1);

LCDWriteData(0X30);

LCDWriteIndex(0xE8);

LCDWriteData(0x85);

LCDWriteData(0x00);

LCDWriteData(0x78);

LCDWriteIndex(0xEA);

LCDWriteData(0x00);

LCDWriteData(0x00);

LCDWriteIndex(0xED);

LCDWriteData(0x64);

LCDWriteData(0x03);

LCDWriteData(0X12);

LCDWriteData(0X81);

LCDWriteIndex(0xF7);

LCDWriteData(0x20);

LCDWriteIndex(0xC0); //Power control

LCDWriteData(0x23); //VRH[5:0]

LCDWriteIndex(0xC1); //Power control

LCDWriteData(0x10); //SAP[2:0];BT[3:0]

LCDWriteIndex(0xC5); //VCM control

LCDWriteData(0x3e); //¶Ô±È¶Èµ÷½Ú

LCDWriteData(0x28);

LCDWriteIndex(0xC7); //VCM control2

LCDWriteData(0x86); //--

LCDWriteIndex(0x36); // Memory Access Control

#ifdef H\_VIEW

LCDWriteData(0x28); //C8 //48 68ÊúÆÁ//28 E8 ºáÆÁ

#else

LCDWriteData(0x48);

#endif

LCDWriteIndex(0x3A);

LCDWriteData(0x55);

LCDWriteIndex(0xB1);

LCDWriteData(0x00);

LCDWriteData(0x18);

LCDWriteIndex(0xB6); // Display Function Control

LCDWriteData(0x08);

LCDWriteData(0x82);

LCDWriteData(0x27);

LCDWriteIndex(0xF2); // 3Gamma Function Disable

LCDWriteData(0x00);

LCDWriteIndex(0x26); //Gamma curve selected

LCDWriteData(0x01);

LCDWriteIndex(0xE0); //Set Gamma

LCDWriteData(0x0F);

LCDWriteData(0x31);

LCDWriteData(0x2B);

LCDWriteData(0x0C);

LCDWriteData(0x0E);

LCDWriteData(0x08);

LCDWriteData(0x4E);

LCDWriteData(0xF1);

LCDWriteData(0x37);

LCDWriteData(0x07);

LCDWriteData(0x10);

LCDWriteData(0x03);

LCDWriteData(0x0E);

LCDWriteData(0x09);

LCDWriteData(0x00);

LCDWriteIndex(0XE1); //Set Gamma

LCDWriteData(0x00);

LCDWriteData(0x0E);

LCDWriteData(0x14);

LCDWriteData(0x03);

LCDWriteData(0x11);

LCDWriteData(0x07);

LCDWriteData(0x31);

LCDWriteData(0xC1);

LCDWriteData(0x48);

LCDWriteData(0x08);

LCDWriteData(0x0F);

LCDWriteData(0x0C);

LCDWriteData(0x31);

LCDWriteData(0x36);

LCDWriteData(0x0F);

LCDWriteIndex(0x11); //Exit Sleep

Delay(120);

LCDWriteIndex(0x29); //Display on

LCDWriteIndex(0x2c);

LCDClear(WHITE);

}

void LCDBGEnable(u8 Enable)

{

GPIO\_WriteBit(LCDCTRL,LCDBG,Enable?Bit\_SET:Bit\_RESET);

}

void LCDSetRegion(u16 x\_start,u16 y\_start,u16 x\_end,u16 y\_end)

{

LCDWriteIndex(0x2a);

LCDWriteData16Bit(x\_start);

LCDWriteData16Bit(x\_end-1);

LCDWriteIndex(0x2b);

LCDWriteData16Bit(y\_start);

LCDWriteData16Bit(y\_end-1);

LCDWriteIndex(0x2c);

}

void LCDDrawPoint(u16 x,u16 y,u16 Data)

{

LCDWriteIndex(0x2a);

LCDWriteData16Bit(x);

LCDWriteIndex(0x2b);

LCDWriteData16Bit(y);

LCDWriteIndex(0x2c);

LCDWriteData16Bit(Data);

}

void LCDFill(u16 x\_start,u16 y\_start,u16 x\_end,u16 y\_end,u16 Color)

{

unsigned int i,j;

LCDSetRegion(x\_start,y\_start,x\_end,y\_end);

GPIO\_ResetBits(LCDCTRL,LCDCS);

GPIO\_SetBits(LCDCTRL,LCDRS);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_16b);

for(i=y\_start;i<y\_end;i++)

{

for(j=x\_start;j<x\_end;j++)

{

SPI\_WriteDat(Color);

}

}

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_8b);

GPIO\_SetBits(LCDCTRL,LCDCS);

LCDSetRegion(0,0,X\_MAX\_PIXEL,Y\_MAX\_PIXEL);

}

void LCDClear(u16 Color)

{

LCDFill(0,0,X\_MAX\_PIXEL,Y\_MAX\_PIXEL,Color);

}

void LCDBitblt(u16 x\_start,u16 y\_start,u16 x\_end,u16 y\_end,u16\* source)

{

unsigned int i,j;

LCDSetRegion(x\_start,y\_start,x\_end,y\_end);

GPIO\_ResetBits(LCDCTRL,LCDCS);

GPIO\_SetBits(LCDCTRL,LCDRS);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_16b);

for(i=y\_start;i<y\_end;i++)

{

for(j=x\_start;j<x\_end;j++)

{

SPI\_WriteDat(\*(source++));

}

}

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_8b);

GPIO\_SetBits(LCDCTRL,LCDCS);

LCDSetRegion(0,0,X\_MAX\_PIXEL,Y\_MAX\_PIXEL);

}

void LCDDrawText(u16 x, u16 y, u16 fc, u16 bc, const char \*s)

{

unsigned short x0=x,y0=y;

if(!s)

{

return;

}

for(;\*s;s++)

{

char chr=\*s;

int i=0;

if(chr=='\r')

{

x0=x;

y0+=16;

if(\*(s+1)=='\n')

{

s++;

}

continue;

}

if(chr>32)

{

chr-=32;

}

else

{

chr=0;

}

LCDSetRegion(x0,y0,x0+8,y0+16);

GPIO\_ResetBits(LCDCTRL,LCDCS);

GPIO\_SetBits(LCDCTRL,LCDRS);

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_16b);

for(;i<16;i++)

{

int j=0;

for(;j<8;j++)

{

if(asc16[chr\*16+i]&(0x80>>j))

{

SPI\_WriteDat(fc);

}

else

{

SPI\_WriteDat(bc);

}

}

}

SPI\_DataSizeConfig(SPI1,SPI\_DataSize\_8b);

GPIO\_SetBits(LCDCTRL,LCDCS);

LCDSetRegion(0,0,X\_MAX\_PIXEL,Y\_MAX\_PIXEL);

x0+=8;

}

}

***Main.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "SysTick.h"

#include "LCD.h"

#include "Composer.h"

#include "Delay.h"

#include "KeyScan.h"

#include "KeyDisp.h"

#include "Auto.h"

#include "SpecialKeyScan.h"

#include "VolDisp.h"

#include "WaveSel.h"

int main()

{

SystemInit();

SysTickInit();

LCDInit();

LCDClear(BLACK);

LCDBGEnable(1);

LCDDrawText(0,0,WHITE,BLACK,"Initializing Composer");

InitComposer();

LCDDrawText(0,16,WHITE,BLACK,"Initializing KeyScan");

KeyScanInit();

SpecialKeyScanInit();

SetVol(255);

LCDDrawText(0,32,WHITE,BLACK,"Initialize Complete");

Delay(300);

SetAllKeyDispDisable();

RefreshKeyDisp();

VolDisp(255);

LCDDrawText(10,10,GREEN,GRAY2,"Left=AutoPlay");

LCDDrawText(10,30,GREEN,GRAY2,"Right=Change wave type");

while(1)

{

u8\* KeyVals=KeyScanGetVals();

u8 i;

u8 Ctl=SpecialKeyGetCtrl();

u8 Ctrl=SpecialKeyGetCtrl();

signed short Enc=SpecialKeyGetEnc();

if(Ctrl & 1)

{

SetAllTuneDisable();

SetAllKeyDispDisable();

DispSongList();

RefreshKeyDisp();

VolDisp(GetVol());

LCDDrawText(10,10,GREEN,GRAY2,"Left=AutoPlay");

LCDDrawText(10,30,GREEN,GRAY2,"Right=Change wave type");

}

if(Ctrl & 2)

{

SetAllTuneDisable();

SetAllKeyDispDisable();

WaveSel();

RefreshKeyDisp();

VolDisp(GetVol());

LCDDrawText(10,10,GREEN,GRAY2,"Left=AutoPlay");

LCDDrawText(10,30,GREEN,GRAY2,"Right=Change wave type");

}

if(Enc)

{

signed int Vol=GetVol()+Enc\*5;

if(Vol>255)

{

Vol=255;

}

if(Vol<0)

{

Vol=0;

}

SetVol(Vol);

VolDisp(Vol);

}

for(i=0;i<36;i++)

{

if(SetTuneEnable(i,KeyVals[i]))

{

SetKeyDisp(i,KeyVals[i]);

}

}

}

}

***Menu.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "LCD.h"

#include "Menu.h"

#include "SpecialKeyScan.h"

#define MaxScrPos (Y\_MAX\_PIXEL/18)

#define MenuTextColor BLACK

#define MenuBackgroundColor WHITE

#define MenuTextColorSelected BLACK

#define MenuBackgroundColorSelected BLUE

void EnterMenu(const struct MenuStruct\* Struct)

{

const struct MenuStruct\* StartMenu=Struct;

u8 ScrPos=0;

u8 i;

u8 Refresh=1;

while(SpecialKeyGetCtrl());

while(1)

{

u8 Ctrl=SpecialKeyGetCtrl();

signed short Enc=SpecialKeyGetEnc();

if(Ctrl & 1)

{

while(SpecialKeyGetCtrl());

return;

}

if(Ctrl & 2)

{

while(SpecialKeyGetCtrl());

(StartMenu+ScrPos)->Callback((StartMenu+ScrPos)->Tag);

while(SpecialKeyGetCtrl());

Refresh=1;

}

if(Enc)

{

Refresh=1;

if(Enc>0)

{

for(;Enc!=0;Enc--)

{

ScrPos++;

if(!(StartMenu+ScrPos)->Callback)

{

ScrPos--;

break;

}

if(ScrPos==MaxScrPos)

{

ScrPos=MaxScrPos-1;

if(!(StartMenu+MaxScrPos)->Callback)

{

break;

}

StartMenu++;

}

}

}

else

{

for(;Enc!=0;Enc++)

{

ScrPos--;

if(ScrPos==(u8)-1)

{

ScrPos=0;

if(StartMenu==Struct)

{

break;

}

StartMenu--;

}

}

}

}

if(Refresh)

{

u8 NoMoreItems=0;

for(i=0;i<MaxScrPos;i++)

{

u16 Text=i==ScrPos?MenuTextColorSelected:MenuTextColor;

u16 Background=i==ScrPos?MenuBackgroundColorSelected:MenuBackgroundColor;

LCDFill(0,18\*i,X\_MAX\_PIXEL,18\*i+18,Background);

if(!NoMoreItems)

{

if((StartMenu+i)->Callback)

{

LCDDrawText(10,18\*i+1,Text,Background,(StartMenu+i)->Name);

}

else

{

NoMoreItems=1;

}

}

}

Refresh=0;

}

}

}

***ScriptPhase.c***

#include "Device/Include/stm32f10x.h" // Device header

#include <stdlib.h>

#include <string.h>

#include "Composer.h"

#include "KeyDisp.h"

#define u32 unsigned int

#define u8 unsigned char

#define IsNum(x) ((x)>='0' && (x)<='9')

#define ToNum(x) ((x)-'0')

const char\* Script = NULL;

u32 Counter = 0;

const char\* Jumps[10];

u8 Last = 0;

signed char Rise = 0;

signed char CRaise = 0;

u8 Delayed=1;

void LoadScript(const char\* NewScript)

{

Script = NewScript;

Counter = 0;

memset(Jumps, NULL, sizeof(Jumps));

Rise = 0;

CRaise = 0;

Delayed=1;

}

u8 GetNote(u8 Num, signed char CRaise, signed char Rise)

{

if (Num)

{

short nn;

if (Num < 4)

{

nn = Num \* 2 +CRaise\* 12+Rise - 2;

}

else

{

nn= Num \* 2 +CRaise\* 12+Rise - 3;

}

if (nn < 0)

{

nn = 0;

}

return (u8)nn;

}

else

{

return 0xFF;

}

}

u32 ScriptTick(void)

{

u8 CurNum = 0;

u8 NoExit = 0;

if (Script == NULL)

{

return 0;

}

if (Counter--)

{

return 1;

}

if(!Delayed)

{

SetTuneEnable(Last, 0);

SetKeyDisp(Last, 0);

Delayed=1;

Counter=0;

return 1;

}

Delayed=0;

if (\*Script == 0)

{

Counter = 0;

return 2;

}

do

{

NoExit = 0;

if (IsNum(\*Script))

{

CurNum = ToNum(\*Script);

Counter = 32;

Rise = 0;

CRaise = 0;

}

else if (\*Script == '\n')

{

if (CurNum == 0)

{

Counter = 1;

Script++;

}

break;

}

else

{

switch (\*Script)

{

case 'X':

case 'x':

Counter = 2;

break;

case '=':

Counter /= 2;

break;

case '.':

Counter += Counter / 2;

break;

case '~':

Counter \*= 2;

break;

case '+':

CRaise++;

break;

case '-':

CRaise--;

break;

case 'b':

case 'B':

Rise--;

break;

case '#':

Rise++;

break;

case 'l':

case 'L':

Script++;

if (IsNum(\*Script))

{

Jumps[ToNum(\*Script)] = Script + 1;

NoExit = 1;

break;

}

else

{

return 0;

}

case 'g':

case 'G':

Script++;

if (IsNum(\*Script))

{

if (Jumps[ToNum(\*Script)])

{

const char\* Dest = Jumps[ToNum(\*Script)];

Jumps[ToNum(\*Script)] = 0;

Script = Dest;

continue;

}

break;

}

else

{

return 0;

}

case ' ':

case '\r':

break;

default:

return 0;

}

}

Script++;

} while (\*Script && (!IsNum(\*Script) || NoExit));

if (CurNum==0 && \*Script==0)

{

return 2;

}

Last = GetNote(CurNum, CRaise,Rise);

SetKeyDisp(Last, 1);

SetTuneEnable(Last, 1);

return 1;

}

***SpecialKeyScan.c***

#include "Device/Include/stm32f10x.h" // Device header

void SpecialKeyScanGPIOInit(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO | RCC\_APB2Periph\_GPIOC, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_6 | GPIO\_Pin\_7 | GPIO\_Pin\_10 | GPIO\_Pin\_11;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IPU;

GPIO\_Init(GPIOC, &GPIO\_InitStructure);

GPIO\_PinRemapConfig(GPIO\_FullRemap\_TIM3,ENABLE);

GPIO\_SetBits(GPIOC, GPIO\_Pin\_6 | GPIO\_Pin\_7 | GPIO\_Pin\_10 | GPIO\_Pin\_11);

}

void SpecialKeyScanTimerInit(void)

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_ICInitTypeDef TIM\_ICInitStruct;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, ENABLE);

TIM\_TimeBaseStructure.TIM\_ClockDivision=TIM\_CKD\_DIV1;

TIM\_TimeBaseStructure.TIM\_CounterMode=TIM\_CounterMode\_Up;

TIM\_TimeBaseStructure.TIM\_Period=0xFFFF;

TIM\_TimeBaseStructure.TIM\_Prescaler=1;

TIM\_TimeBaseStructure.TIM\_RepetitionCounter=0;

TIM\_TimeBaseInit(TIM3, &TIM\_TimeBaseStructure);

TIM\_EncoderInterfaceConfig(TIM3, TIM\_EncoderMode\_TI1, TIM\_ICPolarity\_Falling, TIM\_ICPolarity\_Falling);

TIM\_ICInitStruct.TIM\_Channel=TIM\_Channel\_1;

TIM\_ICInitStruct.TIM\_ICFilter=0xF;

TIM\_ICInitStruct.TIM\_ICPolarity=TIM\_ICPolarity\_Falling;

TIM\_ICInitStruct.TIM\_ICPrescaler=TIM\_ICPSC\_DIV1;

TIM\_ICInitStruct.TIM\_ICSelection=TIM\_ICSelection\_DirectTI;

TIM\_ICInit(TIM3, &TIM\_ICInitStruct);

TIM\_SetCounter(TIM3,0);

TIM\_Cmd(TIM3, ENABLE);

}

void SpecialKeyScanInit(void)

{

SpecialKeyScanGPIOInit();

SpecialKeyScanTimerInit();

}

signed short SpecialKeyGetEnc(void)

{

u16 CounterVal=TIM\_GetCounter(TIM3);

if(CounterVal)

{

TIM\_SetCounter(TIM3,0);

return CounterVal;

}

return 0;

}

u8 SpecialKeyGetCtrl(void)

{

return (~(GPIO\_ReadInputData(GPIOC)>>10)) & 3;

}

***stm32f10x\_it.c***

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file Project/STM32F10x\_StdPeriph\_Template/stm32f10x\_it.c

\* @author MCD Application Team

\* @version V3.5.0

\* @date 08-April-2011

\* @brief Main Interrupt Service Routines.

\* This file provides template for all exceptions handler and

\* peripherals interrupt service routine.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS

\* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE

\* TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY

\* DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING

\* FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE

\* CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

\*

\* <h2><center>&copy; COPYRIGHT 2011 STMicroelectronics</center></h2>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* Includes ------------------------------------------------------------------\*/

#include "Device/Include/stm32f10x.h" // Device header

#include "SysTick.h"

#include "Composer.h"

/\*\* @addtogroup STM32F10x\_StdPeriph\_Template

\* @{

\*/

/\* Private typedef -----------------------------------------------------------\*/

/\* Private define ------------------------------------------------------------\*/

/\* Private macro -------------------------------------------------------------\*/

/\* Private variables ---------------------------------------------------------\*/

/\* Private function prototypes -----------------------------------------------\*/

/\* Private functions ---------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Cortex-M3 Processor Exceptions Handlers \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @brief This function handles NMI exception.

\* @param None

\* @retval None

\*/

void NMI\_Handler(void)

{

NVIC\_SystemReset();

}

/\*\*

\* @brief This function handles Hard Fault exception.

\* @param None

\* @retval None

\*/

void HardFault\_Handler(void)

{

/\* Go to infinite loop when Hard Fault exception occurs \*/

while (1)

{

NVIC\_SystemReset();

}

}

/\*\*

\* @brief This function handles Memory Manage exception.

\* @param None

\* @retval None

\*/

void MemManage\_Handler(void)

{

/\* Go to infinite loop when Memory Manage exception occurs \*/

while (1)

{

NVIC\_SystemReset();

}

}

/\*\*

\* @brief This function handles Bus Fault exception.

\* @param None

\* @retval None

\*/

void BusFault\_Handler(void)

{

/\* Go to infinite loop when Bus Fault exception occurs \*/

while (1)

{

NVIC\_SystemReset();

}

}

/\*\*

\* @brief This function handles Usage Fault exception.

\* @param None

\* @retval None

\*/

void UsageFault\_Handler(void)

{

/\* Go to infinite loop when Usage Fault exception occurs \*/

while (1)

{

NVIC\_SystemReset();

}

}

/\*\*

\* @brief This function handles SVCall exception.

\* @param None

\* @retval None

\*/

void SVC\_Handler(void)

{

}

/\*\*

\* @brief This function handles Debug Monitor exception.

\* @param None

\* @retval None

\*/

void DebugMon\_Handler(void)

{

}

/\*\*

\* @brief This function handles PendSVC exception.

\* @param None

\* @retval None

\*/

void PendSV\_Handler(void)

{

}

/\*\*

\* @brief This function handles SysTick Handler.

\* @param None

\* @retval None

\*/

void SysTick\_Handler(void)

{

SysTickIncreasement();

}

/\*\*

\* @brief This function handles DMA2 interrupt request.

\* @param None

\* @retval None

\*/

void DMA2\_Channel3\_IRQHandler(void)

{

if(DMA\_GetITStatus(DMA2\_IT\_HT3))

{

DMA\_ClearITPendingBit(DMA2\_IT\_HT3);

ConmosePart(1);

}

if(DMA\_GetITStatus(DMA2\_IT\_TC3))

{

DMA\_ClearITPendingBit(DMA2\_IT\_TC3);

ConmosePart(0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* STM32F10x Peripherals Interrupt Handlers \*/

/\* Add here the Interrupt Handler for the used peripheral(s) (PPP), for the \*/

/\* available peripheral interrupt handler's name please refer to the startup \*/

/\* file (startup\_stm32f10x\_xx.s). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @brief This function handles PPP interrupt request.

\* @param None

\* @retval None

\*/

/\*void PPP\_IRQHandler(void)

{

}\*/

/\*\*

\* @}

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT 2011 STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/

***SysTick.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "RTE\_Device.h" // Keil::Device:Startup

volatile u32 Ticks;

void SysTickInit(void)

{

NVIC\_SetPriority(SysTick\_IRQn,NVIC\_EncodePriority(NVIC\_GetPriorityGrouping(),0,0));

Ticks=0;

SysTick\_Config(RTE\_SYSCLK / 10000);

}

void SysTickIncreasement(void)

{

Ticks++;

}

u32 GetSysTick(void)

{

return Ticks;

}

***VolDisp.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "LCD.h"

#define TextColor GREEN

#define TextBackgroundColor GRAY2

#define FrameColor BLACK

#define BackgroundColor GRAY1

#define SlideColor BLUE

#define TextX 3

#define TextY 100

#define SlideX (TextX+35)

#define SlideY (TextY-2)

#define SlideWidth 20

#define SlideHeight 20

#define BorderWidth 2

void VolDisp(u8 Vol)

{

LCDDrawText(TextX,TextY,TextColor,TextBackgroundColor,"VOL:");

LCDFill(SlideX+BorderWidth,SlideY,SlideX+BorderWidth+255+SlideWidth,SlideY+BorderWidth,FrameColor);

LCDFill(SlideX+BorderWidth,SlideY+SlideHeight-BorderWidth,SlideX+BorderWidth+255+SlideWidth,SlideY+SlideHeight,FrameColor);

LCDFill(SlideX,SlideY,SlideX+BorderWidth,SlideY+SlideHeight,FrameColor);

LCDFill(SlideX+BorderWidth+255+SlideWidth,SlideY,SlideX+BorderWidth\*2+255+SlideWidth,SlideY+SlideHeight,FrameColor);

LCDFill(SlideX+BorderWidth,SlideY+BorderWidth,SlideX+BorderWidth+Vol,SlideY+SlideHeight-BorderWidth,BackgroundColor);

LCDFill(SlideX+BorderWidth+Vol+SlideWidth,SlideY+BorderWidth,SlideX+BorderWidth+255+SlideWidth,SlideY+SlideHeight-BorderWidth,BackgroundColor);

LCDFill(SlideX+BorderWidth+Vol,SlideY+BorderWidth,SlideX+BorderWidth+Vol+SlideWidth,SlideY+SlideHeight-BorderWidth,SlideColor);

}

***WaveSel.c***

#include "Device/Include/stm32f10x.h" // Device header

#include "Composer.h"

#include "Menu.h"

#include "Delay.h"

void MenuCallbackWaveSel(u32 Tag);

const struct MenuStruct WaveList[]={

"Sine",&MenuCallbackWaveSel,0,

"Square",&MenuCallbackWaveSel,1,

"Triangle",&MenuCallbackWaveSel,2,

"SawTooth",&MenuCallbackWaveSel,3,

"DualSine",&MenuCallbackWaveSel,4,

0,0,0,

};

void MenuCallbackWaveSel(u32 Tag)

{

SetWaveType(Tag);

SetTuneEnable(0,1);

Delay(100);

SetTuneEnable(0,0);

}

void WaveSel(void)

{

EnterMenu(WaveList);

}

***Auto.h***

void DispSongList(void);

***Composer.h***

void InitComposer(void);

void ConmosePart(u8 FirstPart);

void Compose(void);

u8 SetTuneEnable(u8 Index,u8 Enable);

void SetAllTuneDisable(void);

void SetVol(u8 Vol);

u8 GetVol(void);

void SetWaveType(u8 Type);

***Delay.h***

void Delay(u32 nTime);

void PreciseDelay(u32 nTime);

void Delay100us(u32 nTime);

***FONT.h***

const unsigned char asc16[]={

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //" "

0x00,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,0x10,0x10,0x00,0x00, //"!"

0x00,0x00,0x6C,0x6C,0x24,0x24,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00, //"""

0x00,0x24,0x24,0x24,0x24,0xFE,0x48,0x48,0x48,0x48,0xFC,0x90,0x90,0x90,0x90,0x00, //"#"

0x00,0x10,0x3C,0x54,0x92,0x90,0x50,0x38,0x14,0x12,0x12,0x92,0x54,0x78,0x10,0x00, //"$"

0x00,0x00,0x22,0x5C,0x94,0xA8,0x48,0x10,0x10,0x24,0x2A,0x52,0x54,0x88,0x00,0x00, //"%"

0x00,0x00,0x30,0x48,0x48,0x50,0x20,0x6E,0x54,0x94,0x8C,0x88,0x8A,0x74,0x00,0x00, //"&"

0x00,0x00,0x30,0x30,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //"'"

0x00,0x04,0x08,0x10,0x10,0x20,0x20,0x20,0x20,0x20,0x20,0x10,0x10,0x08,0x04,0x00, //"("

0x00,0x80,0x40,0x20,0x20,0x10,0x10,0x10,0x10,0x10,0x10,0x20,0x20,0x40,0x80,0x00, //")"

0x00,0x00,0x00,0x00,0x10,0x54,0x38,0x10,0x38,0x54,0x10,0x00,0x00,0x00,0x00,0x00, //"\*"

0x00,0x00,0x00,0x10,0x10,0x10,0x10,0xFE,0x10,0x10,0x10,0x10,0x00,0x00,0x00,0x00, //"+"

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x10,0x20,0x00, //","

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFE,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //"-"

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x10,0x00,0x00, //"."

0x00,0x00,0x04,0x04,0x08,0x08,0x10,0x10,0x20,0x20,0x40,0x40,0x80,0x80,0x00,0x00, //"/"

0x00,0x00,0x38,0x44,0x82,0x82,0x82,0x82,0x82,0x82,0x82,0x82,0x44,0x38,0x00,0x00, //"0"

0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00, //"1"

0x00,0x00,0x38,0x44,0x82,0x82,0x04,0x08,0x10,0x20,0x40,0x82,0x84,0xFC,0x00,0x00, //"2"

0x00,0x00,0x38,0x44,0x82,0x02,0x04,0x38,0x04,0x02,0x02,0x82,0x44,0x38,0x00,0x00, //"3"

0x00,0x00,0x04,0x0C,0x14,0x14,0x24,0x24,0x44,0x44,0xFE,0x04,0x04,0x0E,0x00,0x00, //"4"

0x00,0x00,0xFC,0x80,0x80,0x80,0xB8,0xC4,0x82,0x02,0x02,0x82,0x84,0x78,0x00,0x00, //"5"

0x00,0x00,0x3C,0x42,0x82,0x80,0xB8,0xC4,0x82,0x82,0x82,0x82,0x44,0x38,0x00,0x00, //"6"

0x00,0x00,0x7E,0x42,0x82,0x04,0x04,0x08,0x08,0x08,0x10,0x10,0x10,0x10,0x00,0x00, //"7"

0x00,0x00,0x38,0x44,0x82,0x82,0x44,0x38,0x44,0x82,0x82,0x82,0x44,0x38,0x00,0x00, //"8"

0x00,0x00,0x38,0x44,0x82,0x82,0x82,0x82,0x46,0x3A,0x02,0x82,0x44,0x38,0x00,0x00, //"9"

0x00,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x00, //":"

0x00,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x10,0x10,0x20,0x00,0x00, //";"

0x00,0x00,0x00,0x00,0x06,0x18,0x60,0x80,0x60,0x18,0x06,0x00,0x00,0x00,0x00,0x00, //"<"

0x00,0x00,0x00,0x00,0x00,0x00,0xFE,0x00,0x00,0xFE,0x00,0x00,0x00,0x00,0x00,0x00, //"="

0x00,0x00,0x00,0x00,0xC0,0x30,0x0C,0x02,0x0C,0x30,0xC0,0x00,0x00,0x00,0x00,0x00, //">"

0x00,0x38,0x44,0x82,0x82,0x02,0x04,0x08,0x10,0x10,0x10,0x00,0x10,0x10,0x00,0x00, //"?"

0x00,0x00,0x38,0x44,0x82,0x9A,0xAA,0xAA,0xAA,0xAA,0xAA,0x96,0x80,0x42,0x3C,0x00, //"@"

0x00,0x00,0x10,0x10,0x10,0x28,0x28,0x28,0x44,0x44,0x7C,0x44,0x44,0xEE,0x00,0x00, //"A"

0x00,0x00,0xFC,0x42,0x42,0x42,0x42,0x7C,0x42,0x42,0x42,0x42,0x42,0xFC,0x00,0x00, //"B"

0x00,0x00,0x3C,0x44,0x82,0x80,0x80,0x80,0x80,0x80,0x82,0x82,0x44,0x38,0x00,0x00, //"C"

0x00,0x00,0xF8,0x44,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x44,0xF8,0x00,0x00, //"D"

0x00,0x00,0xFC,0x44,0x42,0x40,0x44,0x7C,0x44,0x40,0x40,0x42,0x44,0xFC,0x00,0x00, //"E"

0x00,0x00,0xFC,0x44,0x42,0x40,0x44,0x7C,0x44,0x40,0x40,0x40,0x40,0xF0,0x00,0x00, //"F"

0x00,0x00,0x34,0x4C,0x82,0x80,0x80,0x80,0x8E,0x84,0x84,0x84,0x4C,0x34,0x00,0x00, //"G"

0x00,0x00,0xEE,0x44,0x44,0x44,0x44,0x7C,0x44,0x44,0x44,0x44,0x44,0xEE,0x00,0x00, //"H"

0x00,0x00,0x7C,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00, //"I"

0x00,0x00,0x3E,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x88,0x88,0x70,0x00,0x00, //"J"

0x00,0x00,0xEE,0x44,0x48,0x48,0x50,0x60,0x50,0x48,0x48,0x44,0x44,0xEE,0x00,0x00, //"K"

0x00,0x00,0xE0,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x42,0x44,0xFC,0x00,0x00, //"L"

0x00,0x00,0xC6,0x44,0x6C,0x6C,0x6C,0x54,0x54,0x54,0x44,0x44,0x44,0xEE,0x00,0x00, //"M"

0x00,0x00,0xCE,0x44,0x64,0x64,0x64,0x54,0x54,0x4C,0x4C,0x4C,0x44,0xE4,0x00,0x00, //"N"

0x00,0x00,0x38,0x44,0x82,0x82,0x82,0x82,0x82,0x82,0x82,0x82,0x44,0x38,0x00,0x00, //"O"

0x00,0x00,0xF8,0x44,0x42,0x42,0x42,0x44,0x78,0x40,0x40,0x40,0x40,0xE0,0x00,0x00, //"P"

0x00,0x00,0x38,0x44,0x82,0x82,0x82,0x82,0x82,0x82,0x82,0xBA,0x44,0x3C,0x02,0x00, //"Q"

0x00,0x00,0xF0,0x48,0x44,0x44,0x44,0x48,0x70,0x48,0x44,0x44,0x44,0xE6,0x00,0x00, //"R"

0x00,0x00,0x3C,0x44,0x82,0x80,0x40,0x30,0x0C,0x02,0x02,0x82,0x44,0x78,0x00,0x00, //"S"

0x00,0x00,0x7C,0x54,0x92,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x38,0x00,0x00, //"T"

0x00,0x00,0xEE,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x38,0x00,0x00, //"U"

0x00,0x00,0xEE,0x44,0x44,0x44,0x44,0x28,0x28,0x28,0x28,0x10,0x10,0x10,0x00,0x00, //"V"

0x00,0x00,0xEE,0x44,0x54,0x54,0x54,0x54,0x54,0x54,0x28,0x28,0x28,0x28,0x00,0x00, //"W"

0x00,0x00,0xEE,0x44,0x44,0x28,0x28,0x10,0x10,0x28,0x28,0x44,0x44,0xEE,0x00,0x00, //"X"

0x00,0x00,0xEE,0x44,0x44,0x28,0x28,0x28,0x10,0x10,0x10,0x10,0x10,0x38,0x00,0x00, //"Y"

0x00,0x00,0x7E,0x44,0x84,0x08,0x08,0x10,0x20,0x20,0x40,0x82,0x84,0xFC,0x00,0x00, //"Z"

0x00,0x1C,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x1C,0x00, //"["

0x00,0x00,0xEE,0x44,0x54,0x54,0xFE,0x54,0x54,0x54,0x28,0x28,0x28,0x28,0x00,0x00, //"\"

0x00,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x70,0x00, //"]"

0x00,0x30,0x48,0x84,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //"^"

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFE,0x00, //"\_"

0x00,0x40,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //"`"

0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x84,0x04,0x7C,0x84,0x84,0x8C,0x76,0x00,0x00, //"a"

0x00,0x00,0xC0,0x40,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x42,0x64,0x58,0x00,0x00, //"b"

0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x44,0x80,0x80,0x80,0x80,0x44,0x38,0x00,0x00, //"c"

0x00,0x00,0x0C,0x04,0x04,0x04,0x34,0x4C,0x84,0x84,0x84,0x84,0x4C,0x36,0x00,0x00, //"d"

0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x84,0x84,0xFC,0x80,0x80,0x84,0x78,0x00,0x00, //"e"

0x00,0x00,0x18,0x24,0x20,0x20,0xF8,0x20,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x00, //"f"

0x00,0x00,0x00,0x00,0x00,0x00,0x3A,0x44,0x44,0x78,0x80,0x7C,0x82,0x82,0x7C,0x00, //"g"

0x00,0x00,0xC0,0x40,0x40,0x40,0x58,0x64,0x44,0x44,0x44,0x44,0x44,0xEE,0x00,0x00, //"h"

0x00,0x00,0x10,0x10,0x00,0x00,0x30,0x10,0x10,0x10,0x10,0x10,0x10,0x38,0x00,0x00, //"i"

0x00,0x00,0x10,0x10,0x00,0x00,0x30,0x10,0x10,0x10,0x10,0x10,0x10,0x90,0x60,0x00, //"j"

0x00,0x00,0xC0,0x40,0x40,0x40,0x5C,0x48,0x50,0x60,0x50,0x48,0x44,0xEE,0x00,0x00, //"k"

0x00,0x00,0x30,0x10,0x10,0x10,0x10,0x11,0x10,0x10,0x10,0x10,0x10,0x39,0x00,0x00, //"l"

0x00,0x00,0x00,0x00,0x00,0x00,0xAC,0xD2,0x92,0x92,0x92,0x92,0x92,0xD6,0x00,0x00, //"m"

0x00,0x00,0x00,0x00,0x00,0x00,0x58,0xE4,0x44,0x44,0x44,0x44,0x44,0xEE,0x00,0x00, //"n"

0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x44,0x82,0x82,0x82,0x82,0x44,0x38,0x00,0x00, //"o"

0x00,0x00,0x00,0x00,0x00,0x00,0xD8,0x64,0x42,0x42,0x42,0x64,0x58,0x40,0xE0,0x00, //"p"

0x00,0x00,0x00,0x00,0x00,0x00,0x36,0x4C,0x84,0x84,0x84,0x4C,0x34,0x04,0x0E,0x00, //"q"

0x00,0x00,0x00,0x00,0x00,0x00,0x6C,0x30,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x00, //"r"

0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x88,0x84,0x60,0x18,0x84,0x44,0x78,0x00,0x00, //"s"

0x00,0x00,0x00,0x20,0x20,0x20,0xF8,0x20,0x20,0x20,0x20,0x20,0x24,0x18,0x00,0x00, //"t"

0x00,0x00,0x00,0x00,0x00,0x00,0xC6,0x42,0x42,0x42,0x42,0x42,0x46,0x3A,0x00,0x00, //"u"

0x00,0x00,0x00,0x00,0x00,0x00,0xEE,0x44,0x44,0x28,0x28,0x28,0x10,0x10,0x00,0x00, //"v"

0x00,0x00,0x00,0x00,0x00,0x00,0xEE,0x44,0x44,0x54,0x54,0x28,0x28,0x28,0x00,0x00, //"w"

0x00,0x00,0x00,0x00,0x00,0x00,0xEE,0x44,0x28,0x10,0x10,0x28,0x44,0xEE,0x00,0x00, //"x"

0x00,0x00,0x00,0x00,0x00,0x00,0xEE,0x44,0x44,0x28,0x28,0x10,0x10,0xA0,0xC0,0x00, //"y"

0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x44,0x88,0x10,0x20,0x42,0x84,0xFC,0x00,0x00, //"z"

0x00,0x0C,0x10,0x10,0x10,0x10,0x10,0x60,0x10,0x10,0x10,0x10,0x10,0x10,0x0C,0x00, //"{"

0x00,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x00, //"|"

0x00,0xC0,0x20,0x20,0x20,0x20,0x20,0x18,0x20,0x20,0x20,0x20,0x20,0x20,0xC0,0x00, //"}"

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x72,0x8C,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //"~"

0x00,

};

***KeyDisp.h***

void RefreshKeyDisp(void);

void SetKeyDisp(u8 Index,u8 Enable);

void SetAllKeyDispDisable(void);

***KeyScan.h***

void KeyScanInit(void);

u8\* KeyScanGetVals(void);

***LCD.h***

#define H\_VIEW

#ifdef H\_VIEW

#define X\_MAX\_PIXEL 320

#define Y\_MAX\_PIXEL 240

#else

#define X\_MAX\_PIXEL 240

#define Y\_MAX\_PIXEL 320

#endif

#define RED 0xf800

#define GREEN 0x07e0

#define BLUE 0x001f

#define WHITE 0xffff

#define BLACK 0x0000

#define YELLOW 0xFFE0

#define GRAY0 0xEF7D //»ÒÉ«0 11101 111011 11101

#define GRAY1 0x8410 //»ÒÉ«1 10000 100000 10000

#define GRAY2 0x4208 //»ÒÉ«2 01000 010000 01000

#define LCDCTRL GPIOD

#define LCDRS GPIO\_Pin\_8 //MCU\_PD8 TFT --RS/DC

#define LCDCS GPIO\_Pin\_9 //MCU\_PD9 TFT --CS

#define LCDRST GPIO\_Pin\_10 //MCU\_PD10 TFT --RST

#define LCDBG GPIO\_Pin\_11 //MCU\_PD11 TFT --LED

#define LCDSCK GPIO\_Pin\_5 //MCU\_PA5 TFT --SCK

#define LCDMISO GPIO\_Pin\_6 //MCU\_PA6 MISO TFT --MISO

#define LCDMOSI GPIO\_Pin\_7 //MCU\_PA7 MOSI TFT --MOSI

void LCDInit(void);

void LCDBGEnable(u8 Enable);

void LCDDrawPoint(u16 x,u16 y,u16 Data);

void LCDFill(u16 x\_start,u16 y\_start,u16 x\_end,u16 y\_end,u16 Color);

void LCDClear(u16 Color);

void LCDBitblt(u16 x\_start,u16 y\_start,u16 x\_end,u16 y\_end,u16\* source);

void LCDDrawText(u16 x, u16 y, u16 fc, u16 bc, const char \*s);

***Menu.h***

typedef void(\*MenuCallback)(u32 Tag);

struct MenuStruct

{

const char\* Name;

MenuCallback Callback;

u32 Tag;

};

void EnterMenu(const struct MenuStruct\* Struct);

***ScriptPhase.h***

void LoadScript(const char\* NewScript);

u32 ScriptTick(void);

***SpecialKeyScan.h***

void SpecialKeyScanInit(void);

signed short SpecialKeyGetEnc(void);

u8 SpecialKeyGetCtrl(void);

***SysTick.h***

void SysTickInit(void);

void SysTickIncreasement(void);

u32 GetSysTick(void);

***VolDisp.h***

void VolDisp(u8 Vol);

***WaveSel.h***

void WaveSel(void);