VIN projekt
WAV player

Matic Pristavnik Vrešnjak

Uvod

Cilj projekta je bilo naredit wav player, ki lahko predvaja wav datoteke in izvaja osnovne operacije kot so, nižanje in višanje zvoka, predvajanje / zaustavitev zvoka oz. datoteke. Za izdelavo predvajalnika sem uporabil:

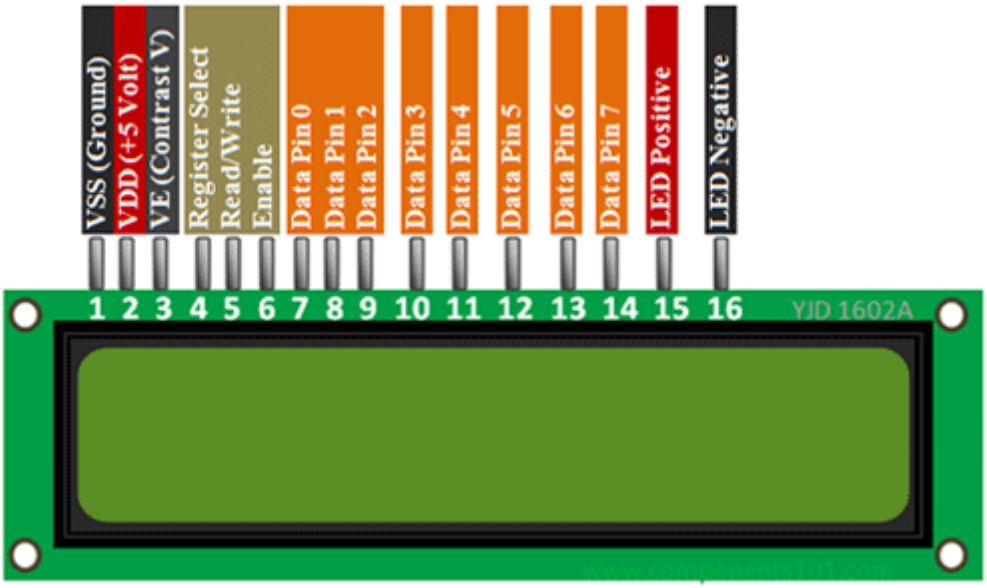
- lCD zaslon 16 * 2
- micro SD card reader
- 4 gumbe

LCD zaslon in stanje predvajalnika

LCD zaslon sem izbral saj sem z njim lažje videl kaj se dogaja v kodi oz. v predvajalniku, prav tako je uporaben za prikaz operacij, ki se izvajajo nad wav datotekami.

Za priključitev LCD zaslona sem uporabil sledečo postavitev:

- PE 7 povezna na rs
- PE 9 povezan na rw
- PE 8 povezan na enable
- PE 10 povezan na D4
- PE 11 povezan na D5
- PE 12 povezan na D6
- PE 13 povezan na D7
- GND povezam na K
- 5V povezan na A



Preko PE 7 oz. rs izbiram kateri registri (ukazni in podatkovni registri) se uporabljajo pri pošiljanju podatkov na LCD zaslon. Ukazne registre uporabljamo takrat, kadar želim spremeniti delovanje zaslona. Za podatkovne registre pa pošljemo podatke za znak oz. niz, ki ga želimo prikazati na zaslonu. Za ukazne registre pošljemo stanje 0 med tem ko za podatkovne pošljemo 1.

PE 9 oz. rw pove ali se podatki berejo ali zapisujejo. Za branje pošljemo 0 med tem ko za zapisovanje pošljemo 1.

PE 8 oz. enable (na LCD označen z E) skrbi za vklapljanje LCD zaslona. 1 je za vklopljen zaslon 0 je za izklopljen.

PE 10 – PE 13 so pini na katere pošiljamo 8-bitne podatke.

K in A sta namenjena za osvetljavo zaslona. zaradi tega ju samo priključimo na GND in 5V za svetlejši zaslon. Z uporabo upornikov ali potenciometra bi lahko spreminjal tudi intenziteto svetlobe.

LCD zaslon je treba pred uporabo treba inicializirati. To storimo tako, da najprej počakamo 50 ms nato pa nastavimo vse pin, da delujejo na RCC uri saj drugače bi lahko povzročali, da se podatki na LCD zaslonu ne prikažejo pravilno. LCD zaslon , ki sem ga uporabil lahko deluje v 4 ali 8 bit načinu. Sam sem nastavil LCD zaslon na 4-bit način. LCD zaslon vklopim tako, da izvedem OR operacijo nad 0x04, 0x00 ter ponovno 0x00. V tem primeru je 0x04 enak LCD_DISPLAYON med tem ko CUSROROFF in BLINKOFF sta 0x00.

```
_displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
```

Nato podatke pošljem kot ukaz na LCD zaslon kjer je nastavim samo rs na 1 oz. izvedem OR operacijo z 0x08 in _displaycontrol.

Potem nastavim oz. pošljem še ukaz kje mora LCD zaslon pričeti prikazovati tekst. Ukaz oblikujem tako, da izvedem OR operacijo na 0x02,0x00 ter 0x04. S tem postopkom je moj LCD zaslon pripravljen na delovanje.

Preostale ukaze, ki jih lahko pošiljamo na LCD zaslon so opisani v spodnji tabeli:

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ
	RS	R/W	DB7	DB6	DB 5	DB4	DB3	DB2	DB 1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

Bolj natančen opis LCD zaslona pa najdete in si ogledate na tej povezavi:

https://components101.com/sites/default/files/component_datasheet/16x2%20LCD%20Datasheet.pdf

Za izpis podatkov moramo nastavit rs pin na 1 ter rw pin na 0. Nato nastavimo še bite za posamezne pine od D4 – D7. S tem izpišemo na LCD zaslon en simbol.

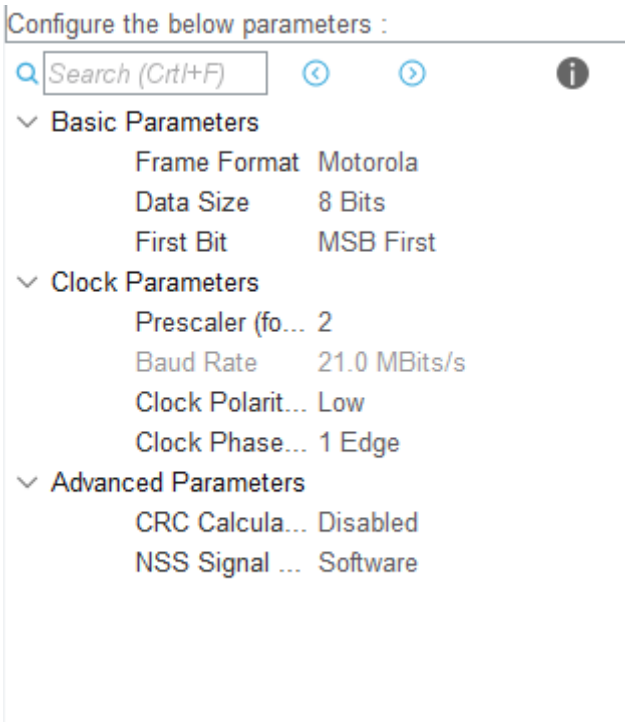
SPI in SD kartica

S pomočjo SPI in micro SD card adapterja sem omogočil branje in zapisovanje podatkov na SD kartico. Najprej sem želel uporabljati SPI1 vendar se je izkazal, da ne deluje pravilno, saj komunicira pri previsoki frekvenci, zato sem uporabil SPI2.

SPI2 pripadajo sledeči pini:

- PC2, ki je MISO
- PC3, ki je MOSI
- PA5, ki je SCK
- PB1, ki je CS

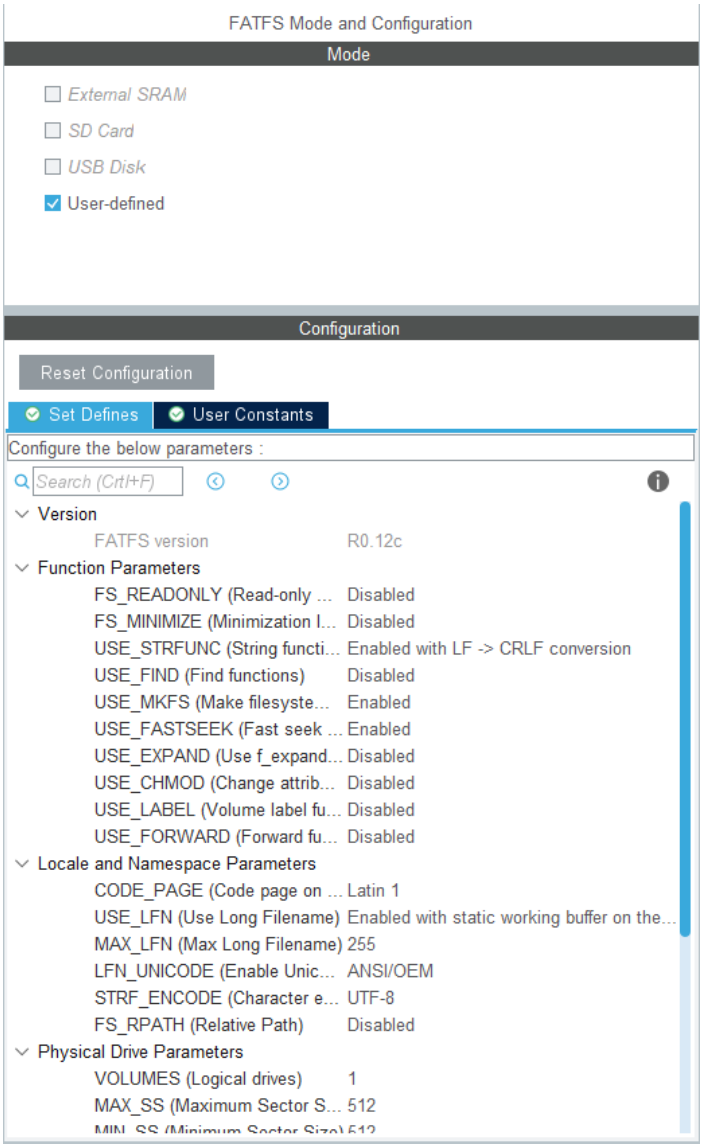
SPI2 je nastavljen, da deluje v full-duplex master načinu med tem ko je Stm32 clock speed nastavljen na 168MHz, kar nam, da sledeče nastavitve z SPI2:



SPI2 in SD card adapter sem povezal na naslednji način.

- PC2 je povezan z MISO
- PC3 je povezan z MOSI
- PA5 je povezan z SCK
- PB1 je povezan z CS
- 5V je povezan z VCC
- GND je povezan z GND na SD adapterju

Ko so bile komponente povezane sem še omogočil FATFS sistem na Stm32. Nastavitve lahko vidimo spodaj:



V primeru, da bi uporabljal SIDO modul za SD kartico bi lahko uporabili že vnaprej pripravljene gonilnike, ker uporabljam SPI sem ji moral narediti sam. Zato sem izbral opcijo User-defined.

Za komunikacijo med SD kartico in mikro krmilnikom uporabljam mmc ukaze. Njihove definicije in oblike so opisane v spodnji tabeli:

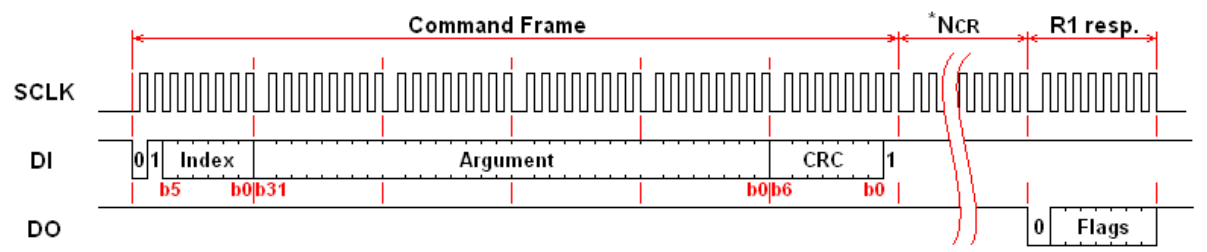
Command Index	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD8	*3	R7	No	SEND_IF_COND	For only SDC V2. Check voltage range.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change R/W block size.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading command of ACMD<n> command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.

*1:ACMD<n> means a command sequense of CMD55-CMD<n>.

*2: Rsv(0)[31], HCS[30], Rsv(0)[29:0]

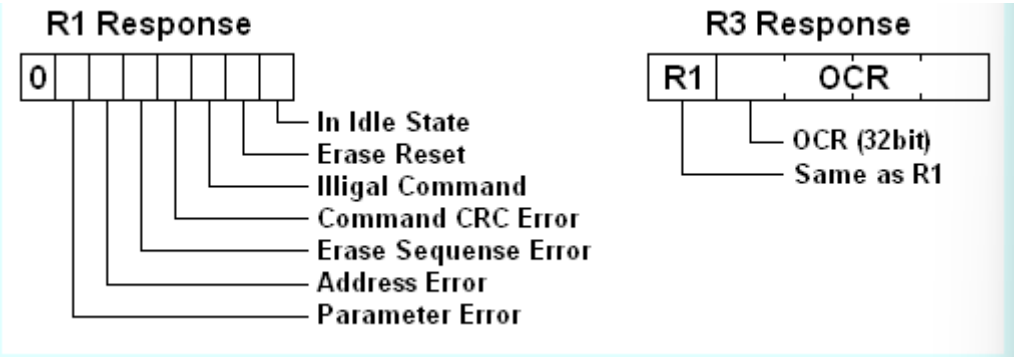
*3: Rsv(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]

Vsak ukaz začnemo najprej z ukaznim indeksom nato podamo argumente, če jih potrebuje ukaz okvir pa zaključimo z CRC biti. Ukazni indeksi imajo vrednosti od 0x40 pa do (0x40+58). Primer sestavitve ukaza lahko vidimo spodaj:



Za vsak ukaz moramo pozneje tudi pričakovati določeno vrsto odgovora odgovori. Vsi odgovori tipa R1 so označeni z MSB bitom na 0x0 v nekaterih primerih pa želimo tudi prejeti 0x1. Pri odgovorih R3 prejmemo iste podatke kot pri R1 plus še vrednost OCR registra.

Opis teh dveh odgovorov sta bolj jasno prikazana spodaj:



Gonilniki za brane SD kartice preko SPI delujejo sledeče:

- Najprej izvedemo inicializacijo kjer preko SPI linije pošljemo nekaj bajtov, da prebudimo SD kartico. Nato pa pošljemo MMC ukaz, ki nastavi SD kartico v čakajoče stanje. Po pošiljanju ukaza čakamo, dokler ne dobimo od SD kartice odgovor oz. število 1, ali pa poteče čas za čakanje na odgovor. Po uspešno prejetem odgovoru, moramo preveriti katere vrste je kartica. V primeru, da je vrsta SD kartice pravilna jo namestimo , med tem ko v primeru, da vrsta kartice ni pravilna jo ugasnemo.

- Za branje podatkov moramo poslati ukaz CMD17, če beremo samo en blok. Če želimo brati več blokov moramo poslati CMD18. V primeru CMD18 moramo poslati še CMD12, ko smo prejeli vse podatke, ki smo jih želeli. Za CMD17 in CMD18 moramo kot argument poslati tudi velikost sektorja, ki ga beremo. Podatke shranimo v buffer. V primeru, da beremo več blokov moramo tudi podati število blokov, ki jih moramo prebrati.

- Za zapisovanje moramo poslati ukaz CMD 24, če zapisujemo en blok. Za zapisovanje več blokov moramo poslati CMD25.

Vsak od zgoraj sledečih opisov je implementiran v svojo funkcijo. Vsako od funkcij je bilo treba vstaviti v datoteko user_diskio.c na sledeče načine:

```
DRESULT USER_read (
    BYTE pdrv,      /* Physical drive nmuber to identi
    BYTE *buff,     /* Data buffer to store read data
    DWORD sector,   /* Sector address in LBA */
    UINT count      /* Number of sectors to read */
)
{
  /* USER CODE BEGIN READ */
    /*return RES_OK;*/
    return SD_disk_read(pdrv, buff, sector, count);
  /* USER CODE END READ */
}

/**
  * @brief  Writes Sector(s)
  * @param  pdrv: Physical drive number (0..)
  * @param  *buff: Data to be written
  * @param  sector: Sector address (LBA)
  * @param  count: Number of sectors to write (1..128)
  * @retval DRESULT: Operation result
  */

DSTATUS USER_initialize (
    BYTE pdrv           /* Physical drive nmuber to id
)
{
  /* USER CODE BEGIN INIT */
    /*
    Stat = STA_NOINIT;
    return Stat;
    */
    return SD_disk_initialize(pdrv);
  /* USER CODE END INIT */
}

#if _USE_WRITE == 1
DRESULT USER_write (
    BYTE pdrv,          /* Physical drive nmuber to id
    const BYTE *buff,   /* Data to be written */
    DWORD sector,       /* Sector address in LBA */
    UINT count          /* Number of sectors to write
)
{
  /* USER CODE BEGIN WRITE */
  /* USER CODE HERE */
    /*return RES_OK;*/
    return SD_disk_write(pdrv, buff, sector, count);
  /* USER CODE END WRITE */
}
#endif /* _USE_WRITE == 1 */
```

V primeru, da gonilniki delujejo pravilno mora funkcij f_mount vrniti FR_OK. To lahko tudi vidimo na LCD zaslonu saj v pravilnem delovanju SD kartice izpiše SD: [OK].

Predvajanje zvoka

Za predvajanje zvoka sem uporabil primere, ki jih je ST dal za krmilnik. Preden želimo uporabljati predlogo, moramo poskrbeti, da imamo pravilne nastavitve krmilnika. Vključiti moramo I2C nato pa moramo vključiti še I2S3. Pri I2S3 je treba še vključiti DMA. Izbral sem še Master clock output in Half-Duplex Master. Izbrana frekvenca vzorčenja je 96 KHz. Prav tako je predlagano, da ima PLLI2S *N nastavljen na X 191 v nastavitvah ure.

Preden želimo predvajati wav datoteko jo moramo najti, zato preden poženemo AUDIO kodek, preiščemo celotni root direktorij in si zapomnimo vse datoteke, ki so tipa wav. Ko najdemo vse wav datoteke, izberemo datoteko, ki jo želimo prebrati. To storimo tako, da povemo kje se nahaja ime datoteke. Iz datoteke preberemo najprej wav header, nato pa v buffer shranim 65536 bajtov iz datoteke. S pomočjo DMA pošljemo prek I2S podatke na napravo za predvajanje zvoka.

Prekinitve in gumbi

Za pravilno delovanje gumbov sem moral uporabiti prekinitve, saj bi se drugače lahko zgodilo, da ne bi v vseh primerih delovali pravilno.

Gumbi so povezani na sledeče pin - e:

- PB11 je nižanje zvoka
- PB12 je višanje zvoka
- PB13 je ena wav datoteka nazaj
- PB14 je ena wav datoteka naprej
- PB0 je stop/play gumb

Nastavitve za pin -e lahko vidite spodaj:

Pi...	Signal...	GPIO ...	GPIO ...	GPIO ...	Maxi...	User ...	Modifi...
PA0-...	n/a	n/a	Exter...	No pu...	n/a		☐
PB0	n/a	Low	Outpu...	No pu...	Low		☐
PB1	n/a	Low	Outpu...	No pu...	Low		☐
PB4	n/a	Low	Outpu...	No pu...	Low		☐
PB11	n/a	n/a	Exter...	Pull-up	n/a		☑
PB12	n/a	n/a	Exter...	Pull-up	n/a		☑
PB13	n/a	n/a	Exter...	Pull-up	n/a		☑
PB14	n/a	n/a	Exter...	Pull-up	n/a		☑

Prav tako pa je bilo potrebno še obkljukati v NVIC opciji:

- EXIT line0 interrupt
- EXIT line[15:10] interrupt

Koda, ki se mora izvesti ob prekinitvi je zapisan v funkciji HAL_GPIO_EXIT_Callback.

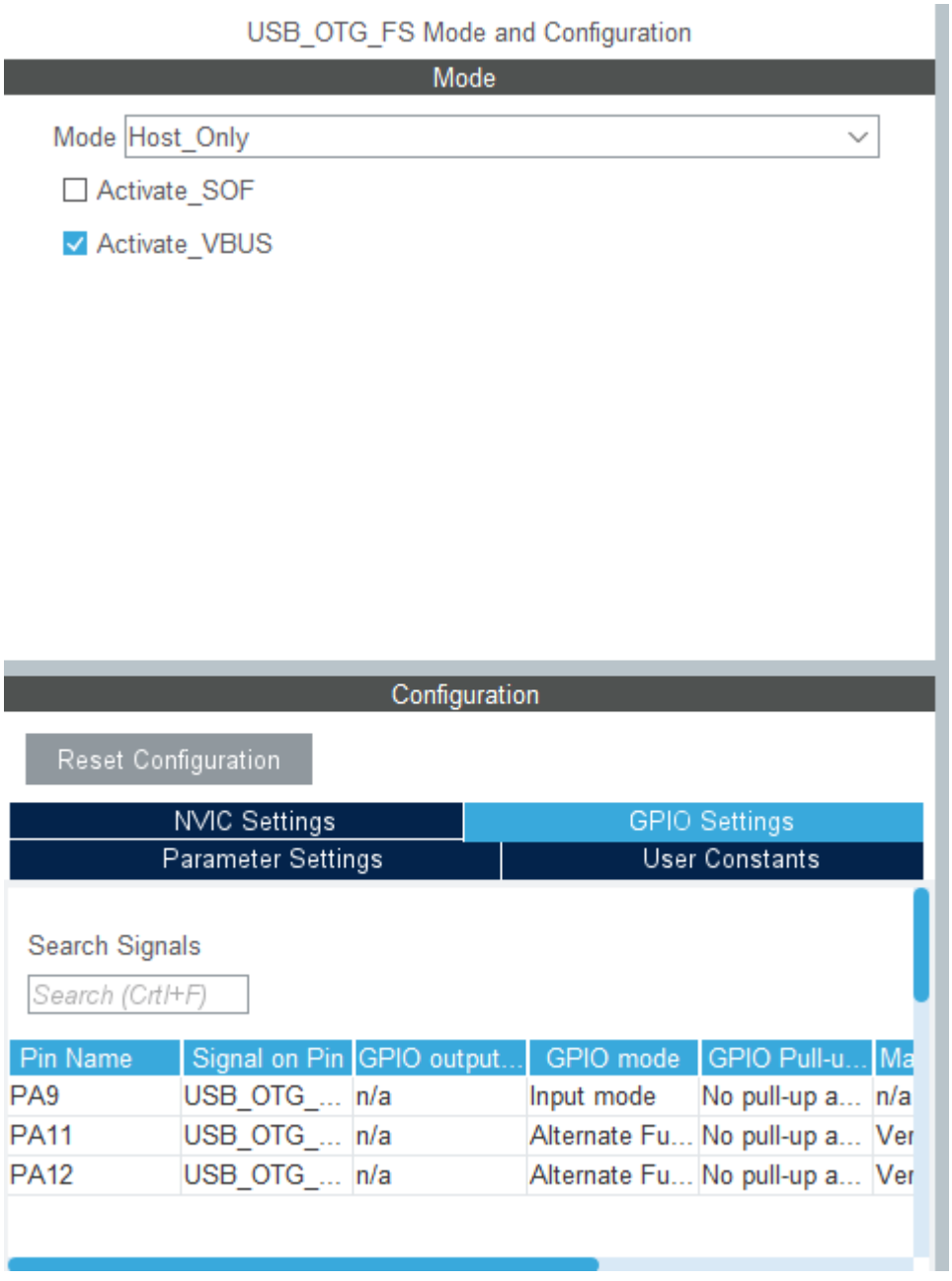
Prehod na USB

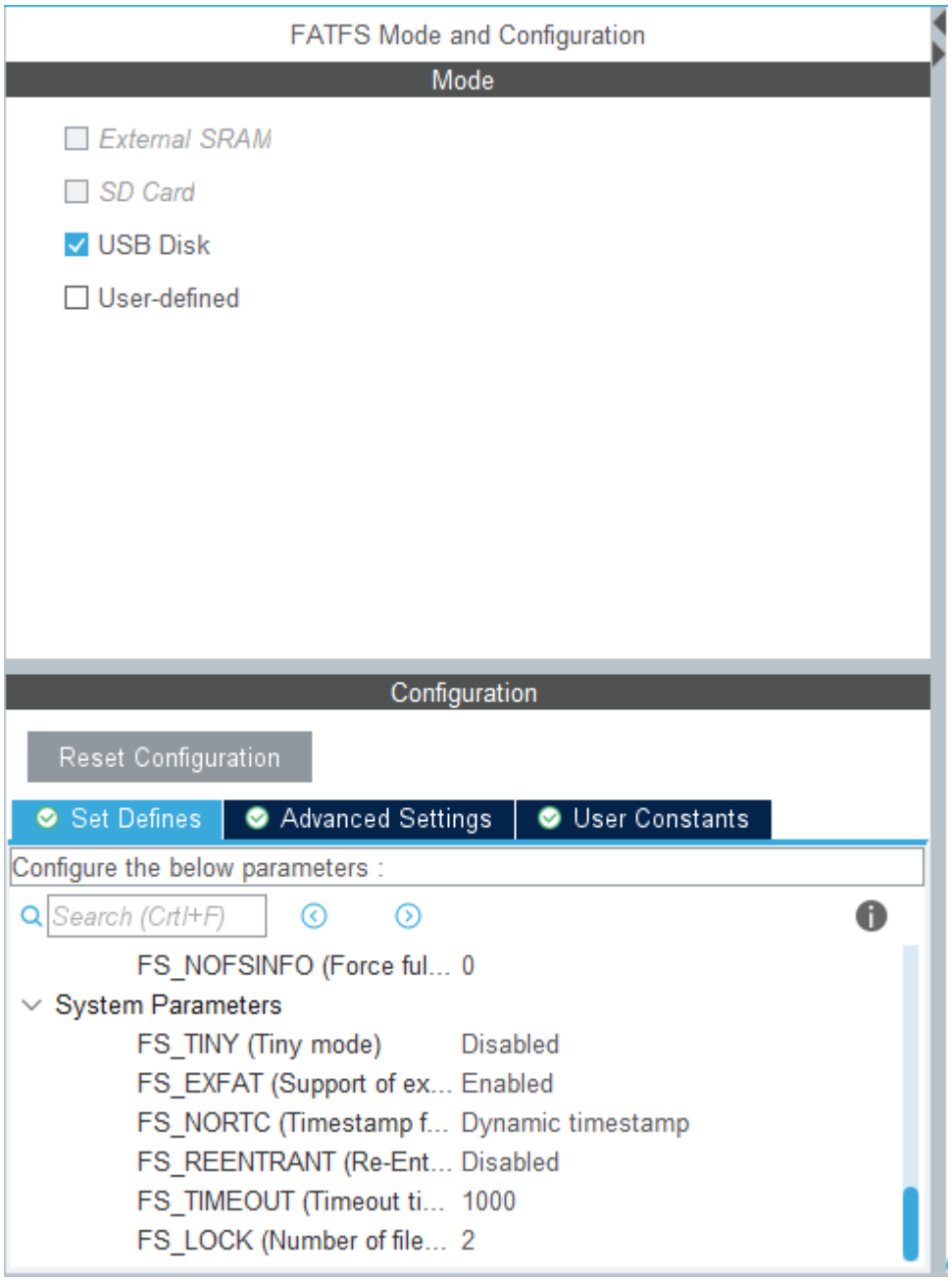
Po implantaciji vseh ključnih delov wav playerja se je izkazalo, da je SPI neprimeren za predvajanje zvoka, saj povzroča prekinitve, ki se zgodijo ob vsakem branju. Zato sem branje podatkov spremenil, da se berejo s USB povezave.

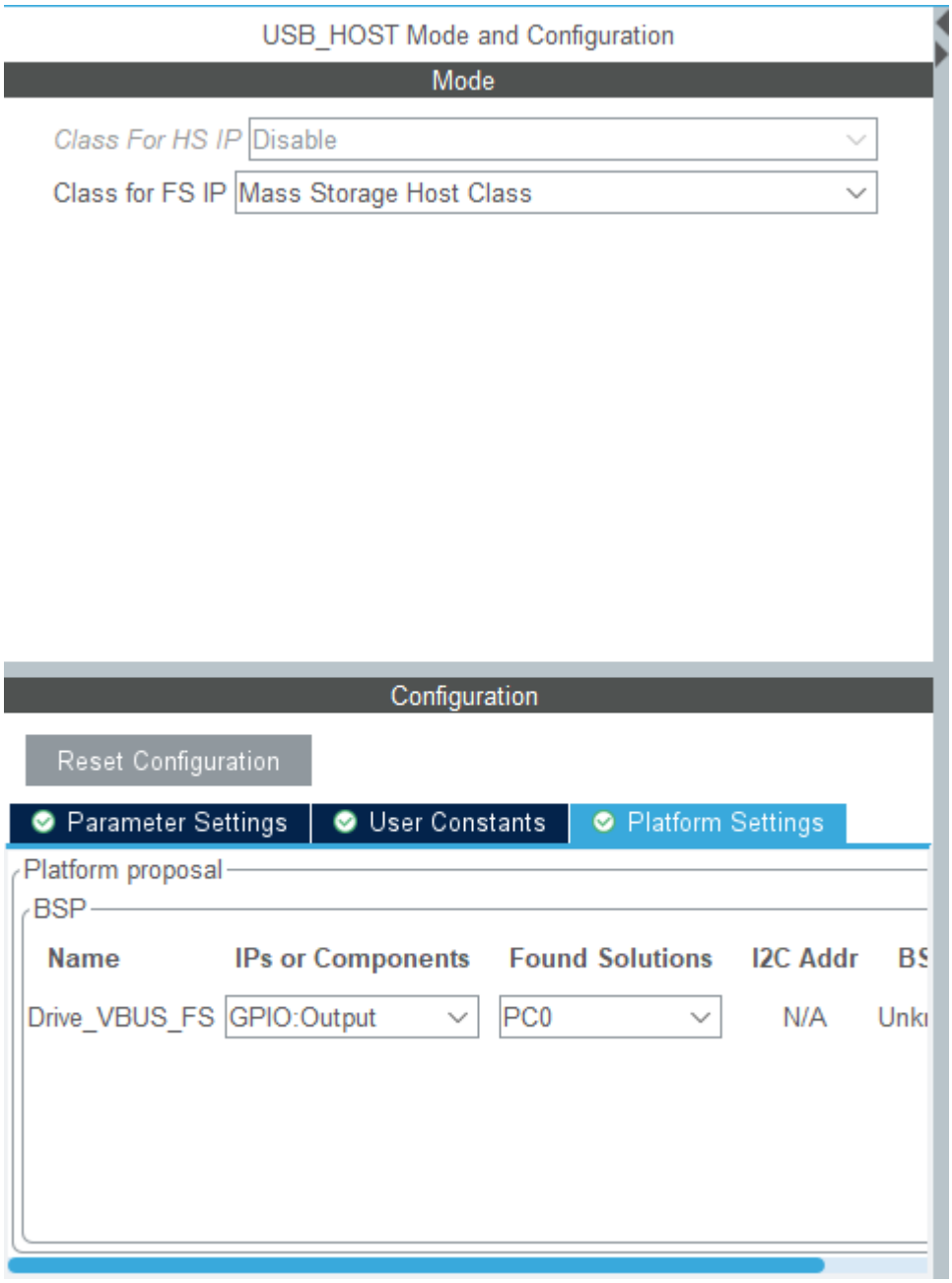
V primeru, da želimo uporabljati USB povezavo moramo spremeniti sledeče:

- Vključiti moramo USB_OTG_FS v Host_Only način, prav tako pa izberemo Active_VBUS.
- Nastaviti moramo USB_HOST razred na Mass Storage Host Class ter za Driver_VBUS_FS izbrati pin PC0 saj bo ta skrbel, da dovaja tok do USB vključka.
- Na koncu še v FATFS izberemo USB Disk namesto User-defind (V primeru, da uporabljamo usb napravo, ki je formatirano z exFAT moramo izbrati še opcijo FS_EXFAT)

Slike nastavitev za USB:




FATFS Mode and Configuration
Mode
External SRAM
SD Card
USB Disk
User-defined
Configuration
Reset Configuration
Set Defines
Advanced Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
FS_NOFSINFO (Force ful... 0
System Parameters
FS_TINY (Tiny mode) Disabled
FS_EXFAT (Support of ex... Enabled
FS_NORTC (Timestamp f... Dynamic timestamp
FS_REENTRANT (Re-Ent... Disabled
FS_TIMEOUT (Timeout ti... 1000
FS_LOCK (Number of file... 2



Gonilniki so že generirani oz. podani preko CUBEIde. V main.c v funkciji sem moral dodati funkcijo , ki preverja stanje USB povezave. Ko je USB naprava pripravljena se namesti kot disk, nato pa se izvede postopek za predvajanje zvoka.

Zaključek

Izvorno kodo in video posnetek lahko najdete nad sledeči povezavi:

Izvorna koda: https://github.com/ColdAlgorithem/WavPlayer

Video: https://youtu.be/6yyBIXG7d9s