

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

ДИПЛОМНА РАБОТА

Тема: Цифров генератор на сигнали с тъчскрийн

Дипломант:
Петър Николов

Научен ръководител:
гл. ас. инж. Любомир Богданов

**С О Ф И Я
2 0 2 0**



Дата на заданието: 15.11.2019 г.
Дата на предаване: 15.02.2020 г.

Утвърждавам:.....
/проф. д-р инж. Т. Василева/

ЗАДАНИЕ за дипломна работа

на ученика Петър Иванов Николов, 12.Б клас

1. Тема: Генератор на сигнали с тъчскрийн

2. Изисквания:

- 2.1. Разработване на принципна електрическа схема и печатна платка на устройството.
- 2.2. Управление на хардуерната система с микроконтролер.
- 2.3. Имплементиране на протокол/драйвер за работа с тъчскрийн LCD.
- 2.4. Имплементиране на протокол за работа с DDS система.
- 2.5. Възможност за параметризиране на характеристиките (форма, амплитуда, честота) на изходния генериран сигнал.

3. Съдържание

- 3.1. Обзор
- 3.2. Изисквания към продукта
- 3.3. Същинска част
- 3.4. Ръководство на потребителя
- 3.5. Заключение

Дипломант :.....
Ръководител:.....
/ гл. ас. д-р инж. Любомир Богданов /
Директор:.....
/ доц. д-р инж. Ст. Стефанова /

Мнение

на научния ръководител за дипломанта Петър Николов

Дипломантът работи усърдно през изминалите месеци върху заданието и успя да покрие всички точки. Реализиран е както софтуерът, така и хардуерът. Отбелязвам, че дипломантът е проектирал и поръчал печатна платка. Според мен дипломантът се е справил отлично!

Дипломантът Петър Николов се е справил отлично с поставената му тема и предлагам да бъде допуснат до защита!

Предлагам за рецензент: Росен Витанов

Дата: 11.03.2020 г.

Ръководител:
гл. ас. д-р инж. Любомир Богданов
ТУ-София

Съдържание

Задание	2
Мнение	3
Съдържание	4
Увод	7

ГЛАВА I: Методи за генериране на произволни сигнали и принципи при управление на вградено устройство с тъчскрийн еcran

1.1. Налични комерсиални продукти	8
1.2. Методи за генериране на произволни сигнали	11
1.2.1. Аналогов генератор на сигнали	11
1.2.2. Цифров генератор на сигнали	11
1.2.3. Сравнение между изброените методи	12
1.3. Взаимодействие с тъчскрийн модул	15
1.3.1. Описание на избрания тъчскрийн модул	15
1.3.2. Управление на LCD екрана на избраната развойна платка	16
1.3.3. Взаимодействие с панела за допир на избраната развойна платка	18

ГЛАВА II: Проектиране на блокова схема на цифров генератор на сигнали и определяне на основните изисквания

2.1. Изисквания към генератора на сигнали	20
2.1.1. Функционални изисквания	20
2.1.2. Електрически изисквания	20
2.1.3. Изисквания към управляващия софтуер	21
2.2. Блокова схема на устройството	22

ГЛАВА III: Проектиране на принципна електрическа схема на цифров генератор на сигнали

3.1. Елементна база и избор на основни интегрални схеми	25
3.1.1. Микроконтролер	25
3.1.2. Статична памет с произволен достъп (SRAM)	26
3.1.3. ЦАП	26
3.1.4. Двоични броячи и логически схеми	27
3.1.5. Програмируем осцилатор	27
3.1.6. Преходник последователна-паралелна комуникация	30
3.1.7. Аналогови интегрални схеми	31
3.1.8. Цифров потенциометър	31
3.2. Принципна електрическа схема	33

3.2.1. Блок „Интерфейс към захранване“	33
3.2.2. Захранване на логически схеми и неизползвани логически елементи	35
3.2.3. Блок „Микроконтролер и допълнителна периферия“	35
3.2.4. Блок „Интерфейс към тъчскрийн модула“	38
3.2.5. Блок „DDS система“	45
3.2.5.1. Очертание „SRAM памет“	45
3.2.5.2. Блок „16-битов адресен брояч“	46
3.2.5.3. Очертание „Програмираме осцилатор“	47
3.2.5.4. Очертание „Преходник последователна-паралелна комуникация“	48
3.2.5.5. Очертание „ЦАП“	48
3.2.6. Блок „Изходна аналогова схемотехника“	50
3.3. I ² C адресация	55

ГЛАВА IV: Проектиране на графични оригинали на печатни платки

4.1. Корпуси на използваните елементи	61
4.2. Параметри на проводящите писти	61
4.3. Особености на слоевете на печатнана платка	65
4.3.1. Сигналните слоеве (1-ви и 4-ти)	65
4.3.2. Захранващият слой (2-ри)	67
4.3.3. Заземяващият слой (3-ти)	68
4.4. Други особености на печатната платка	69
4.4.1. Размери	69
4.4.2. Монтиране	69
4.5. Визуализиране на печатната платка с 3D модели	69

ГЛАВА V: Разработка на софтуерен драйвер за взаимодействие с тъчскрийн модула и алгоритъм за управление на устройството

5.0. Предпоставки	76
5.1. Софтуерна среда за разработка на програмата на микроконтролера	76
5.2. Файлова структура на разработения софтуер	77
5.3. Помощен код за реализиране на управляващата програма	79
5.3.1. Булеви променливи	79
5.3.2. Операции с входно-изходни пинове	80
5.3.3. SPI контролер	81
5.3.4. TWI контролер	83
5.4. Драйвер за управление на тъчскрийн модула	86
5.4.1. Комуникиране с драйверната интегрална схема за LCD экрана	86
5.4.2. Рисуване върху LCD экрана	95
5.4.3. Обработка на информацията, получена от панела за допир	103
5.5. Алгоритъм за управление на устройството	113
5.5.1. Блокова диаграма на управляващия устройството алгоритъм	113
5.5.2. Генериране на електрически сигнал с произволна форма	113
5.5.3. Настройване параметрите на генерирания сигнал	122

ГЛАВА VI: Създаване на работоспособен модел на цифровия генератор на сигнали

6.1. Оживяване на печатната платка на генератора на сигнали	126
6.1.1. Модификации на печатната платка	127
6.1.2. Други проблеми на проектираното устройство	132
6.2. Зареждане на управляващия софтуер	133
6.2.1. Компилиране на управляващата програма	133
6.2.2. Програмиране на микроконтролера ATmega2561	133
6.3. Изследване на изградения работоспособен модел	136
6.3.1. Опитна постановка	136
6.3.2. Резултати от изследванията	136
6.4. Икономическият аспект	141
 Заключение	 143
 Използвани източници	 144
 Приложение	
A. Допълнителни графични и снимкови материали	147

УВОД

Всеки инженер, занимаващ се с електроника, в своята работа се сблъсква с проблеми от най-различно естество при проектиране на електрически схеми, съдържащи компоненти от цифровата или аналоговата схемотехника. При анализиране на действието на изгответи макети на електронни устройства има множество инструменти, с които си служи инженерът - осцилоскоп, спектрален анализатор, мултици и т.н. Един изключително полезен инструмент за инженера е генераторът на сигнали. Последното представлява устройство, което има способността да генерира електрически сигнали с различни форми, както и лесно и удобно да изменя основните му параметри - честота, амплитуда и т.н. Подобен тип генератори се използват в процеса на разработка и тестване на други електронни устройства или части от последните - чрез генератора лесно и прецизно може да се възпроизведе примерен входен сигнал за изследваната електрическа схема и да се анализира изхода ѝ с помощта на други лабораторни инструменти (напр. осцилоскоп).

Разбира се, професионални генератори на сигнали се предлагат на пазара. Тези устройства са много качествени и удобни и притежават способностите да генерират сигнали с произволна форма и да регулират основните им параметри в много голям обхват. Все пак се повдига въпросът дали е възможно от широкодостъпни електронни компоненти да се разработи генератор, който да служи „достатъчно добре“ на потребителя непрофесионалист, макар и да не притежава способностите на комерсиалния продукт. Тази дипломна работа се цели да отговори на поставения въпрос, като предлага оригинална разработка на подобен тип устройство.

Целите на настоящата дипломна работа е да се разработи генератор на сигнали с произволна форма, който да има възможността да изменя сравнително прецизно основните параметри на сигнала (конкретно - честота, амплитуда, постояннотоково отместване) в сравнително големи обхвати. Устройството трябва да предостави интерфейс за взаимодействие с потребителя посредством тъчскрийн течнокристален экран.

ГЛАВА I

Методи за генериране на произволни сигнали и принципи при управление на вградено устройство с тъчскрийн еcran

1.1 Налични комерсиални продукти

Както е упоменато в увода, на пазара съществуват готови комерсиални генератори на сигнали. Някои от наличните пазарни продукти са разгледани и съпоставени в тази глава. Целта на тази точка от дипломната работа не е да рекламира специфични комерсиални продукти, а да запознае читателя с очакваните способности на устройства тип генератор на сигнали.

Първо, нека отбележим, че в индустрията се срещат два термина, които обозначават устройства от типа генератор на сигнали:

- Arbitrary Function Generator (AFG) - генератор на функции. Подобни устройства имат възможността да възпроизведат **само и единствено** сигнали с определени форми (синусоидална, правоъгълна, триъгълна и др.). Функциите (сигналите с форми, описани от математически формули) са записани трайно във вътрешната памет на устройството от производителя и потребителят има възможността да изменя само параметрите на генерирация сигнал (частота, амплитуда и др.).
- Arbitrary Waveform Generator (AWG) - генератор на сигнали с произволна форма. Тези устройства са подобни на генераторите на функции, имайки възможността да възпроизведат математически описана функция под формата на електрически сигнал. Отличителната характеристика на генераторите на сигнали с произволна форма е възможността им да възпроизведат електрически сигнал, който не е предварително програмиран в устройството и е описан от самия потребител. Устройствата от този тип обикновено са по-скъпи, защото изискват по-сложна електроника за възпроизвеждане на сигнал с произволна форма, както и по-сложен потребителски интерфейс, който да предостави на потребителя възможността да опише своя сигнал. Разбира се, тези устройства също имат възможността да изменят основните параметри на генерирания сигнал без значение дали последният е програмирана от производителя функция или описан от потребителя.

Фокусът на настоящата дипломна работа е конкретно върху генераторите на сигнали с произволна форма (AWG). Разработеното от дипломанта и описаното в този документ устройство е от такъв тип.

Първият комерсиален продукт, който ще бъде разгледан, е серията SDG800, чийто представител е показан на фиг. 1.1, на SIGLENT Technologies. Устройствата от тази серия имат само един канал, 5 стандартни изходни сигнала, 46 допълнителни вградени сигнали с нестандартни форми, възможност за амплитудна модулация, честотна модулация и др., възможност за управление на устройството чрез USB, възможност за записване на 10 произволни форми на генериран сигнал, описани от потребителя, в енергонезависима памет, максимална честотна лента до 30 MHz и много други способности и характеристики. Всички споменати досега качества на серията SDG800 са описани в каталожните ѝ данни⁽¹⁾. Инструментите от тази серия се предлагат на пазара в диапазона \$239-\$269. Тези, както и инструменти с подобни способности, се считат за „бюджетни решения“, насочени към непрофесионалистите и любителите на електрониката.



фиг. 1.1: Siglent SDG805

Следващият продукт, който е разгледан, също се произвежда от SIGLENT Technologies - серията SDG6000X, чийто представител е показан на фиг. 1.2. Някои от характеристики на серията са следните: два отделни канала за генериране на сигнали, 196 вградени функции, USB интерфейс, LAN интерфейс, максимална честотна лента до 500 MHz и т.н. Подробна информация за серията генератори на сигнали може да се намери в каталожните ѝ данни⁽²⁾. Тези устройства са насочени към специалистите в областта на електрониката, които работят по изключително комплексни проекти. Ценовия диапазон за

серията SDG6000X е \$1499-\$5299, като цената на устройството се определя от максималната му честотна лента.



фиг. 1.2: Siglent SDG6052X

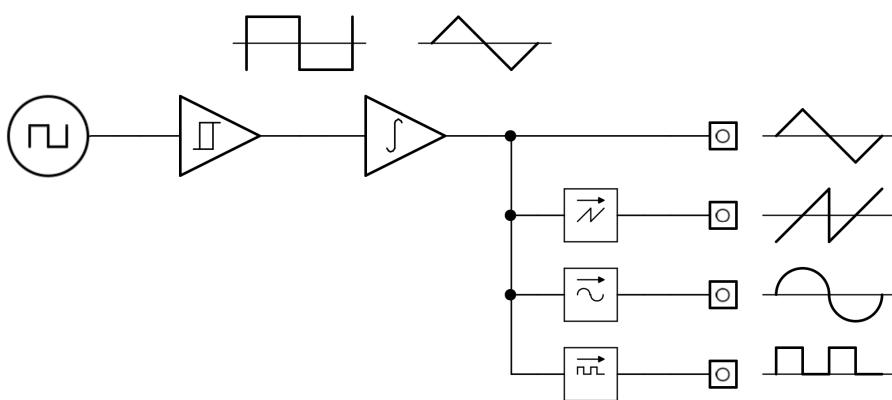
От направеното проучване може да се заключи, че различните комерсиални генератори на сигнали имат най-различни способности и характеристики и служат, както и за целите на простия проект на любителя на електрониката, така и за целите на сложните продукти, разработвани в индустрията. Въпреки това дори и „бюджетните“ генератори на сигнали с произволна форма са сравнително скъпи устройства, които един непрофесионалист в областта на електрониката трудно може да си позволи. Също така много от възможностти на предлаганите лабораторни инструменти остават неупотребени от непрофесионалния потребител. С настоящата дипломна работа дипломантът се цели да разработи „минималистичен“ генератор на сигнали с произволна форма, който да има добри характеристики в сравнение с някои комерсиални продукти и да задоволява нуждите от подобно устройство на инженера непрофесионалист.

1.2 Методи за генериране на произволни сигнали

В тази точка са разгледани накратко и съпоставени два основни метода за генериране на произволни сигнали - на аналогов принцип и на цифров принцип.

1.2.1 Аналогов генератор на сигнали

Този метод използва единствено аналогова схемотехника, за да генерира произволни сигнали. Примерна блокова схема на аналогов генератор на сигнали е представен на фиг. 1.3.

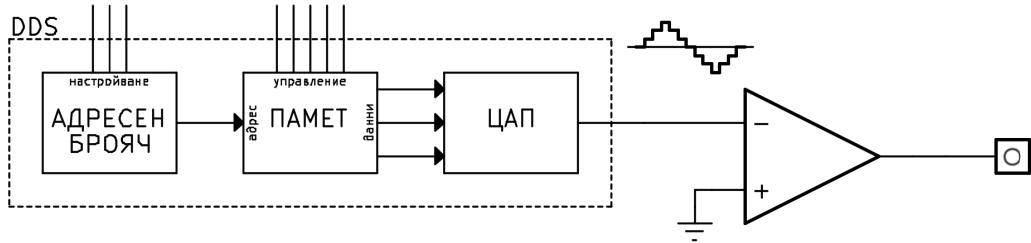


фиг. 1.3: Примерна блокова схема на аналогов генератор на сигнали

Принципът на действие на показаната схема е следният. Правоъгълен сигнал се подава на входа на линеен интегратор, който генерира сигнал с триъгълна форма на изхода си. Този изходен сигнал сам по себе си е напълно използваем и изведен като отделен изход. За постигане на сигнали с други форми (напр. рампова, ШИМ, синусоидална и др.) триъгълният сигнал се оформя от специализиран хардуер, чието действие се описва от определени математически функции.

1.2.2 Цифров генератор на сигнали

Този метод представя даден сигнал като поредица от цифрови данни (числа), които се преобразуват в аналогов сигнал и обработват от изходна аналогова схемотехника. За генериране на сигнала се използва метод, наречена Direct Digital Synthesis (DDS) - Директен Цифров Синтез. Тази техника е обяснена детайлно от Analog Devices в документите (3) и (4). Примерна структура на DDS система е представена на фиг. 1.4.



фиг. 1.4:Примерна структура на DDS система

В центъра на DDS система стоят два основни компонента - памет и ЦАП. В клетките от паметта са записани последователно моментните стойности на сигнал с определена форма за целия му период. Клетките се обхождат последователно от адресен брояч, чиято честота на броене се настройва. Моментната стойност на всяка клетка се преобразува от цифрово-аналогов преобразувател (ЦАП) в аналогово напрежение. Съвкупността от всички получени аналогови нива на напрежение представляват желания генериран аналогов сигнал, който обикновено се обработва (регулира се амплитудата му, усилва се по мощност и т.н.) от допълнителна аналогова схемотехника. Обработеният аналогов сигнал се подава на изхода на устройството за полза от потребителя.

1.2.3 Сравнение между изброените методи

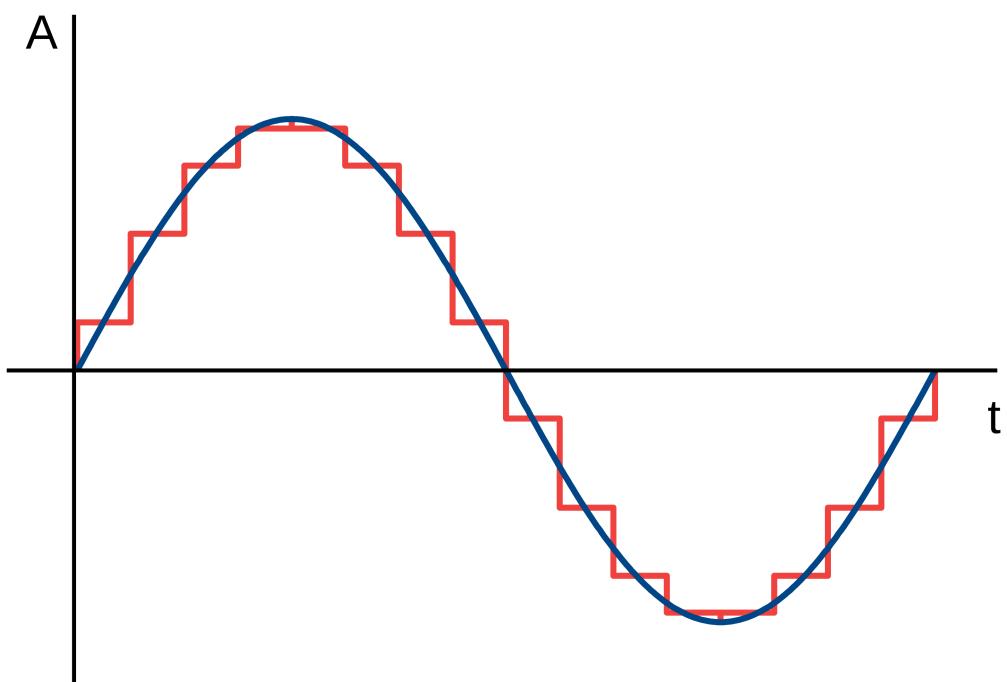
И двета метода се използват за сходни цели - генериране на произволни сигнали, но използват коренно различни подходи. Затова двета разгледани метода имат различни предимства и недостатъци.

Най-голямото предимство на аналогия генератор на сигнали е възпроизвеждането на произволния сигнал само с аналогова електроника. Този метод възпроизвежда желания сигнал с наличието на минимални шумове. Основният недостатък на всички аналогови генератори е нуждата от отделна специализирана схемотехника за възпроизвеждане на всяка отделна форма. Тези устройства могат да генерират само ограничен брой функции и нямат възможността да възпроизведат сигнал с произволна форма. Следователно всички аналогови генератори на сигнали са генератори на функции.

От друга страна, цифровите генератори на сигнали имат възможността да генерират сигнали с произволна форма, ако производителят предостави потребителски интерфейс за въвеждане и записване в паметта на произволна форма. Друго удобство при цифровите генератори е лесното им управление и настройване чрез цифрови интерфейси като

тъчскрийн, USB и др. Но цифровите генератори имат един основен недостатък - дискретизацията на генерираания сигнал. Този феномен е представен нагледно на фиг. 1.5. Причините за дискретизацията са следните:

- Паметта на дадена DDS система е ограничена. Следователно желания генериран аналогов сигнал не може да бъде описан с безкрайно количество моментни стойности за един негов период.
- Резулюцията на цифрово-аналоговия преобразувател определя максималният брой възможни стойности, които могат да се преобразуват от устройството. Но аналогов сигнал (като представения в синьо синусоидален на фиг. 1.5) се описва от безкрайно количество възможни стойности.



фиг. 1.5: Дискретизация на синусоидален сигнал

В резултат на изброените два недостатъка дадена DDS система винаги възпроизвежда стъпаловиден аналогов сигнал (изобразен с червено на фиг. 1.5). Ако двата недостатъка на системата са достатъчно смекчени (паметта има достатъчно капацитет и резулюцията на преобразувателя е достатъчно голяма) изходният генериран стъпаловиден аналогов сигнал се доближава до желания „съвършен“ аналогов сигнал.

В днешно време цифровата техника е развита до степен, в която дискретизацията на желания сигнал е незабележима и не повлиява на работата на потребителя. Затова повечето предлагани на пазара генератори на сигнали се изработват с DDS системи. Въпреки напредването на цифровата схемотехника, има тясно специализирани области в електрониката, където се използват аналогови генератори на сигнали.

Настоящата дипломна работа задълбочава само в областта на директния цифров синтез (DDS), понеже разработеното оригинално устройство използва DDS система.

1.3 Взаимодействие с тъчскрийн модул

За целите на дипломната работа потребителският интерфейс се реализира чрез течноокристален дисплей с панел за допир.

1.3.1 Описание на избрания тъчскрийн модул

В проектираното устройство е използвана развойната платка от фиг. 1.6. Тя включва течноокристален еcran, панел за допир, гнездо за SD карта, както и всички необходими интегрални схеми и допълнителни компоненти за управлението на тъчскрийн модула.



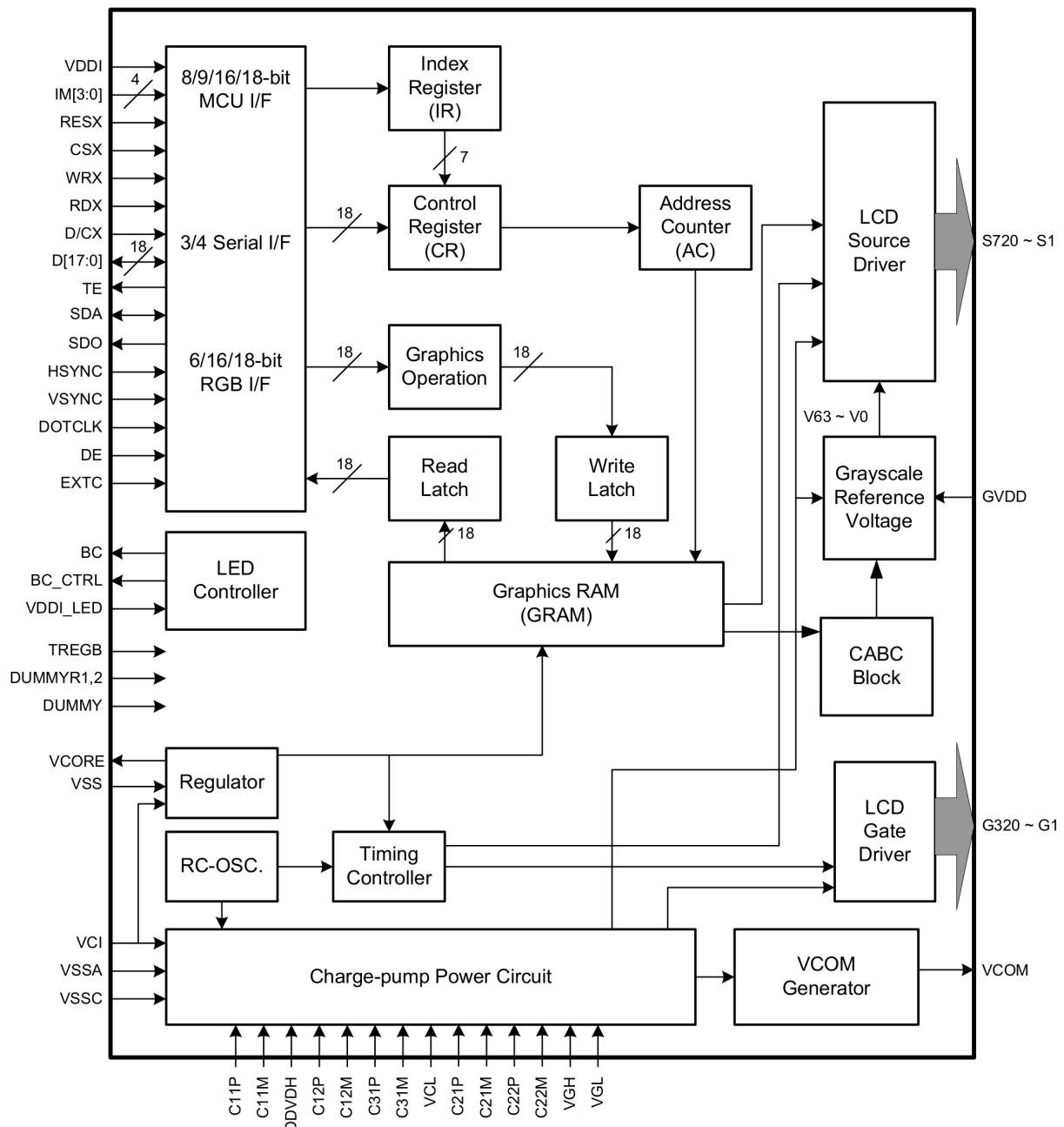
фиг. 1.6: Развойна платка на течноокристален еcran с панел за допир

Дипломантът има за цел да разработи управляващ софтуер за течноокристалния 3.2-инчов дисплей с резолюция 240x320. Гнездото за SD карта не се използва в настоящата дипломна работа и няма да бъде коментирано повече.

Използваният в настоящата дипломна работа тъчскрийн модул е детайлно описан в уебстраницата на LCDWIKI за развойната платка⁽³⁸⁾.

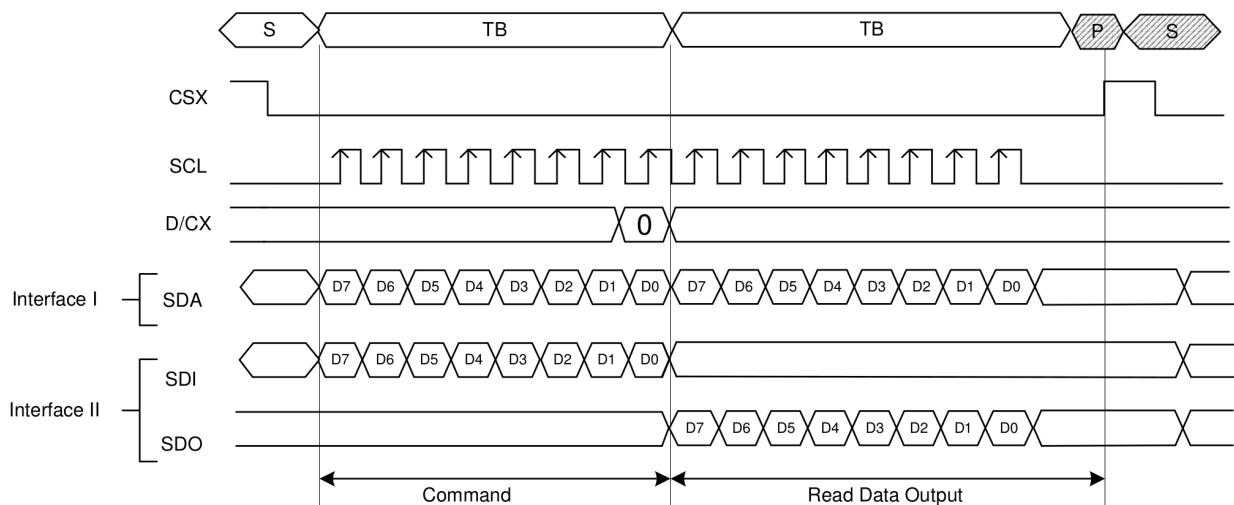
1.3.2 Управление на LCD екрана на избраната развойна платка

Течнокристалният дисплей се управлява от интегралната схема ILI9341⁽²⁵⁾, представляващ драйверен чип за RGB LCD екрани с резолюция 240x320 и с възможност за 262 хил. различни цвята. Блоковата схема на чипа е представена на фиг. 1.7, взета от каталожните му данни⁽²⁵⁾.



фиг. 1.7: Блокова схема на ILI9341

В центъра на чипа е разположен блок с графична RAM памет (GRAM), който има размер 1 382 400 бита (240x18x320 бита). Всяка 18-битова клетка от GRAM паметта описва един пиксел, чийто цвят представлява RGB комбинация, при която и червеният, и зеленият, и синият цвят се описват със 6-битово число. ILI9341 използва съдържанието на графичната памет, за да управлява гейтовете (320 на брой) и сорсовете (720 на брой) на течнокристалния екран. Друга особеност на чипа е наличието на множество цифрови интерфейси за управление на интегралната схема. Налични са паралелни (8-, 9-, 16- и 18-битови), серийни (с 3 или с 4 управляващи линии) и RGB интерфейси (6-, 16- и 18-битови). За управление на ILI9341 дипломантът използва серийния интерфейс с 4 управляващи линии. Другите интерфейси, както и останалата част от вътрешната структура на чипа не са разгледани в настоящата дипломна работа.



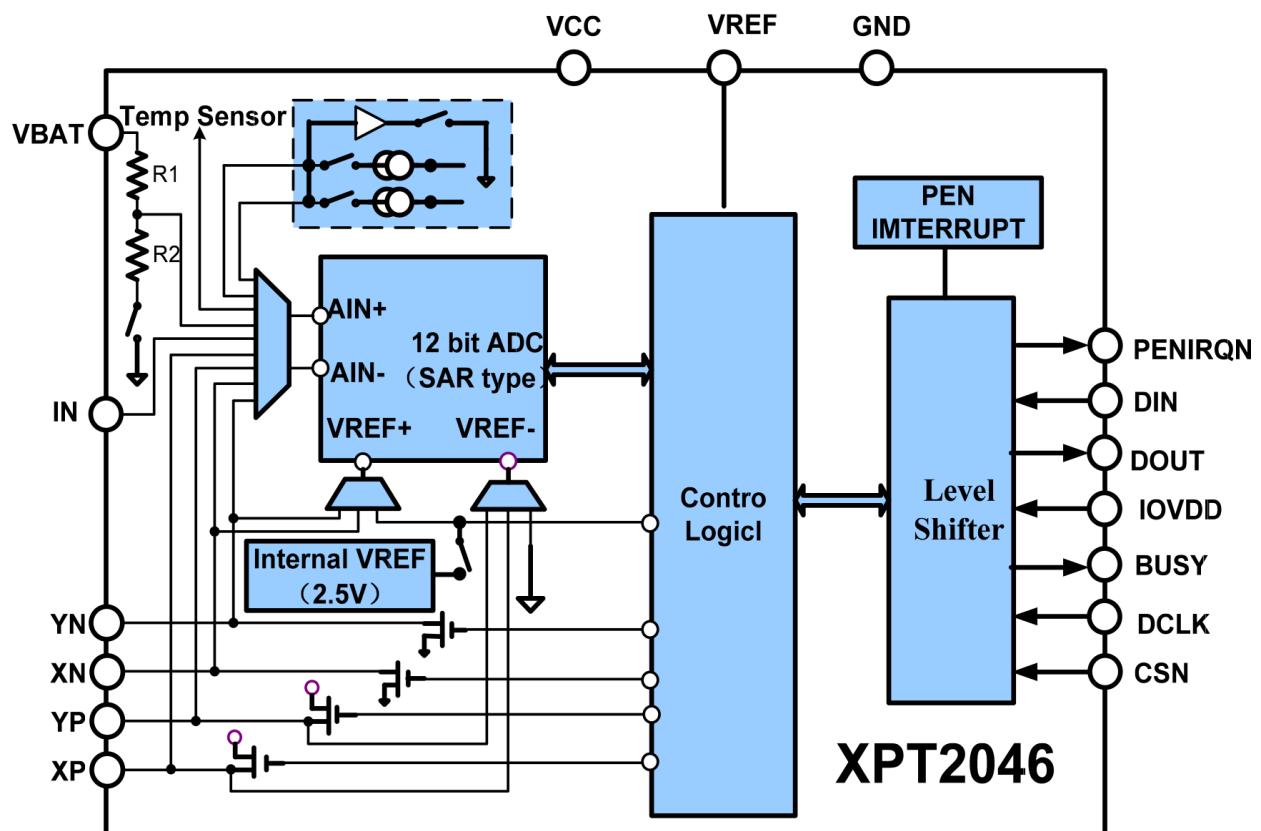
фиг. 1.8: Времева диаграма на серийна комуникация с 4 управляващи линии

На фиг. 1.8 е представена времева диаграма на примерна комуникация с ILI9341, използвайки серийния интерфейс с 4 управляващи линии. За стартиране на комуникацията се подава ниско ниво на CSX (Chip Select) линията. D/CX линията пояснява дали пристигащата информация да се интерпретира като команда за чипа или параметър за дадена команда. На тази линия се подава логическа „0“, след което се изпраща последователно 8-битова команда на SDI линията, започвайки от най-старшия бит. SCL линията се използва за синхронизация на серийната комуникация. След изпращане на командата чипът ILI9341 може или да изпрати обратно информация по SDO линията, или да приеме допълнителни параметри по SDI в

зависимост от типа на обработваната команда (които биват или за четене на данни от ILI9341, или за записване на данни/настройване на чипа).

1.3.3 Взаимодействие с панела за допир на избраната развойна платка

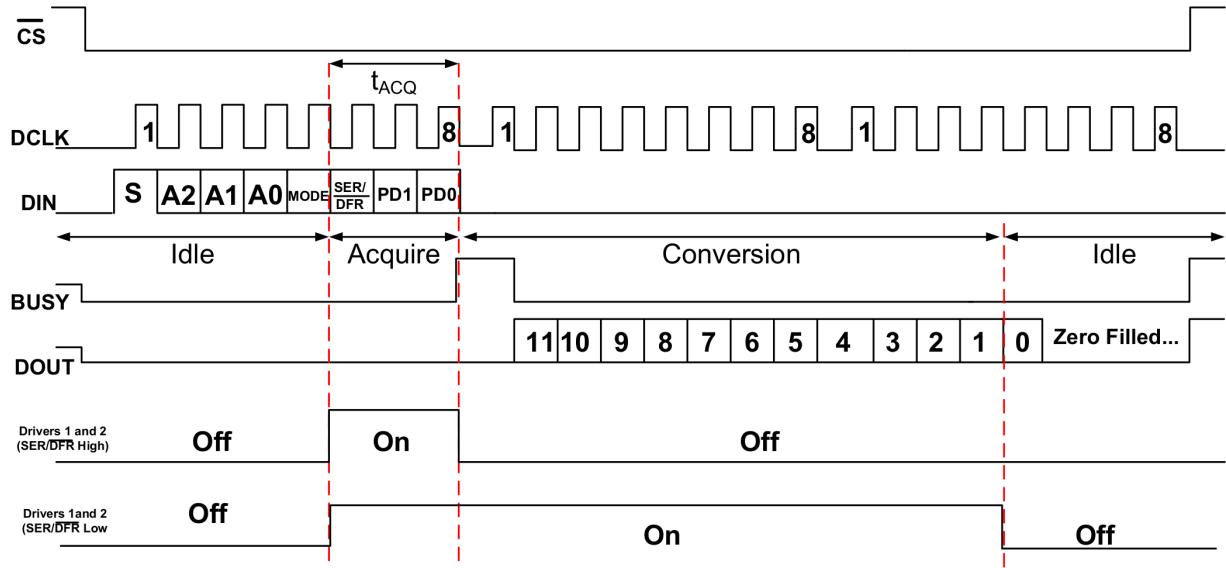
Панелът за допир се управлява от интегралната схема XPT2046⁽²⁶⁾. На фиг. 1.9 е представена блоковата схема на XPT2046. Устройството представлява 12-битов АЦП от тип SAR (Successive Approximation Register) с вътрешен източник на опорно напрежение +2.5V, както и необходимата цифрова логика и интерфейс за управлението му. Чрез извършване на няколко преобразувания от АЦП-то, интегралната схема определя координатите на допирната точка при натискане на панела.



фиг. 1.9: Блокова схема на XPT2046

Интегралната схема XPT2046 се управлява от SPI интерфейс. На фиг. 1.10 е представена времева диаграма на примерна комуникация с XPT2046. За стартиране на комуникацията се подава ниско ниво на \overline{CS} линията. След това следва да се изпрати контролен байт по DIN линията. Подробности относно контролния байт може да се открият в

каталожните данни⁽²⁶⁾ на интегралната схема. След приемане на контролния байт и извършване на преобразуване от АЦП-то чипът изпраща обратно резултата по DOUT линията, започвайки от най-старшия бит. Тъй като резултатът е 12-битов, а са необходими два SPI цикъла (16 такта) за изпращането му, последните (най-младшите) 4 бита са запълнени с логическа „0“, като четещото устройство трябва да ги пренебрегне.



Тъчскрийн контролерът разполага и с допълнителен извод PENIRQN, който е във високо ниво (логическа „1“) в нормално състояние. При докосване на панела за допир интегралната схема генерира прекъсване и изпраща логическа „0“ на този извод.

ГЛАВА II

Проектиране на блокова схема на цифров генератор на сигнали и определяне на основните изисквания

2.1 Изисквания към генератора на сигнали

С настоящата дипломна работа дипломантът се цели да разработи цифров генератор на сигнали с произволна форма с един изходен канал. Устройството трябва да се управлява посредством тъчскрийн дисплей. В тази глава се излагат специфичните изисквания към проектираното устройство, както и неговата блокова схема.

2.1.1 Функционални изисквания

Проектираното устройство трябва да спази следните функционални изисквания:

- Да генерира сигнали с произволни форми.
- Да предостави възможност за изменение на честотата на генерираия сигнал.
- Да предостави възможност за изменение на амплитудата на генерираия сигнал.
- Да предостави възможност за добавяне на постоянноотоково отместване към генерираия сигнал.
- Да осъществи комуникация с външно включен тъчскрийн дисплей.
- Да предостави подходящ потребителски интерфейс посредством периферно включения тъчскрийн модул.

2.1.2 Електрически изисквания

Проектираното устройство трябва да изпълни следните електрически изисквания:

- Да предостави интерфейс за включване на външно захранване.
- Да използва две отделни и несвързани помежду си захранвания - цифрово и аналогово.
 - Цифровата схемотехника на устройството да се захранва от цифровото захранване.
 - Аналоговата схемотехника на устройството да се захранва от аналоговото захранване.

- Да предостави интерфейс за неколкократно програмиране на управляващата програма на устройството.
- Да предостави интерфейс за периферно включване на тъчскрийн модул.
- Да има един изходен канал, който да предостави генерираания сигнал за полза от потребителя.
- Да изменя честотата на генерирания сигнал по електронен път.
- Да изменя амплитудата на генерирания сигнал по електронен път.
- Да изменя постояннотоковото отместване на генерирания сигнал по електронен път.

2.1.3 Изисквания към управляващия софтуер

Разработеният управляващ софтуер за проектираното устройство трябва да спази следните изисквания:

- Да установи успешна комуникация с и да настройва драйверния чип ILI9341, управляващ LCD экрана.
 - Да реализира способия за чертане и рисуване върху течнокристалния экран.
- Да установи успешна комуникация с и да настройва драйверния чип XPT2046, управляващ панела за допир върху экрана.
 - Да обработва информацията, получена от чипа при докосване на экрана.
- Да настройва генератора на сигнали.
 - Да управлява хардуера, който описва произволната форма на генерирания сигнал.
 - Да управлява хардуера, който добавя постоянно отместване към генерирания сигнал.
 - Да управлява хардуера, който променя амплитудата на генерирания сигнал.
 - Да управлява хардуера, който определя честотата на генерирания сигнал.

2.2 Блокова схема на устройството

На фиг. 2.1 е представена блоковата схема на проектираното от дипломанта устройство.

В центъра на генератора стои блокът на DDS системата. Той самият е образуван от няколко отделни компонента, свързани помежду си. Блокът „SRAM“ е съставен от статична памет с произволен достъп, която изпълнява ролята на таблица, съдържаща моментните стойности на генерирания сигнал за целия му период. Паметта се обхожда последователно от блока „Адресен брояч“, съставляващ двоичен брояч с толкова изхода, колкото са адресиращите изводи на интегралната схема тип SRAM. Моментните стойности (под формата на двоични цифрови числа) се подават на следващия компонент - блока „ЦАП“, който преобразува получените стойности в аналогово напрежение с определено ниво. Последователната и периодична промяна на изходното аналогово ниво образува генерирания сигнал с желаната форма на изхода на блока „DDS система“. Скоростта на последователното обхождане на клетките на паметта се контролира от програмируемия осцилатор, който подава контролен сигнал към адресния брояч за увеличаване на стойността си. Следователно честотата на генерирания от DDS системата сигнал се изменя чрез настройване на програмируемия осцилатор.

Възпроизведените от DDS системата аналогов сигнал е изключително маломощен. Затова генеририаният сигнал се подава на блок „Изходна аналогова схемотехника“, който усилва генеририания сигнал по мощност. Също така този блок регулира амплитудата на сигнала и добавя постояннотоково отместване към него. Обработеният генериран сигнал се подава на блок „Изходен канал“, от където потребителят получава годния за употреба генериран сигнал с желаната форма.

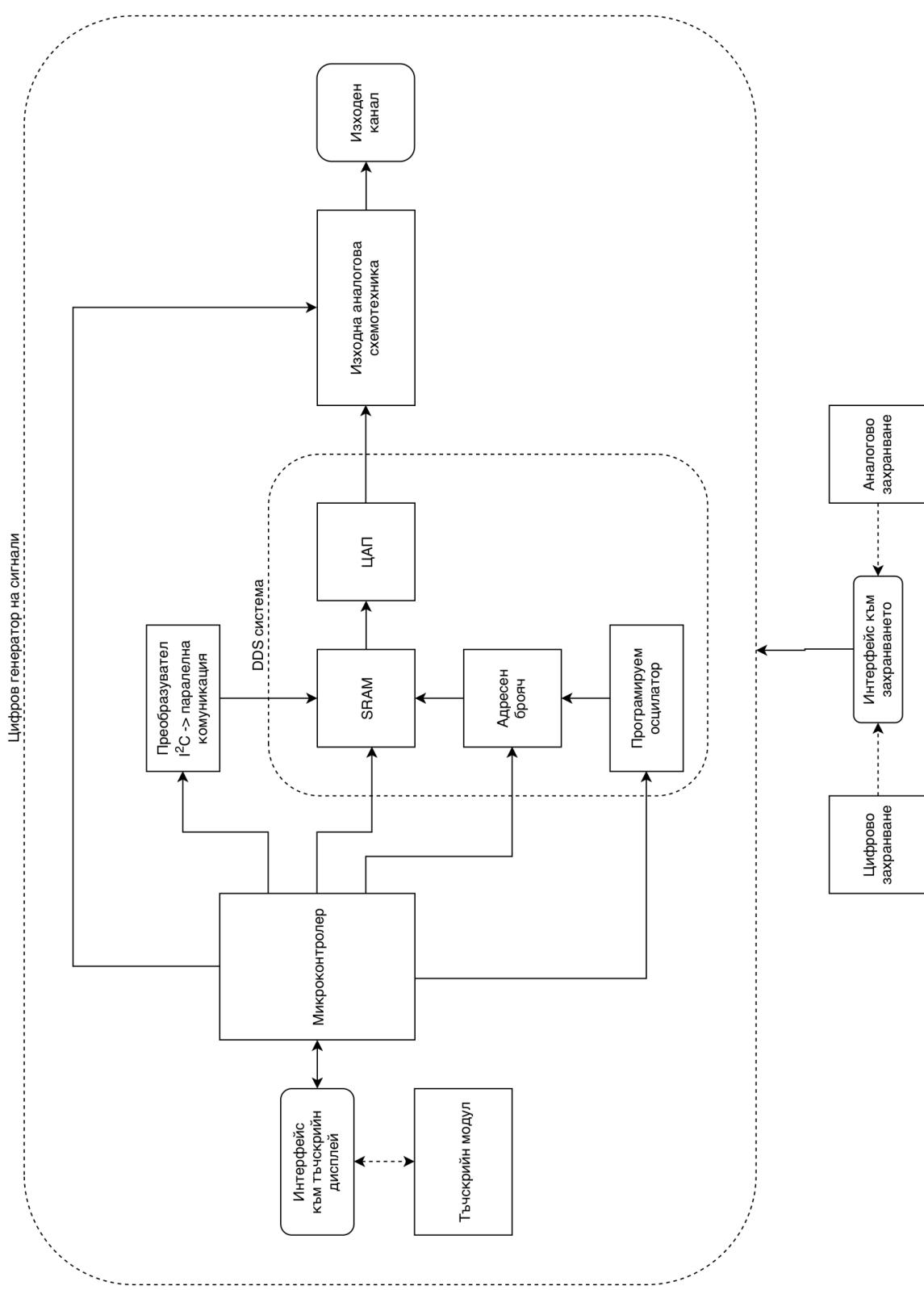
Системата за възпроизвеждане и обработка на желания от потребителя сигнал се настройва и управлява от блока „Микроконтролер“, включващ микроконтролер, както и всичките му необходими периферни елементи за правилното му функциониране. Микроконтролерът изпълнява следните функции:

- Настройване честота на генерирания сигнал чрез настройване на блока „Програмирам осцилатор“.
- Регулиране на амплитудата и постояннотоковото отместване на генерирания сигнал чрез управление на блока „Изходна аналогова схемотехника“.
- Записване на желаната форма на генерирания сигнал чрез записване на моментните стойности в паметта на DDS системата. Презаписването на паметта се постига чрез

последователното обхождане на клетките ѝ от адресния брояч, който получава контролни сигнали директно от микроконтролера, и записване на съответната моментна стойност. За подаване на моментната двоична стойност се използва преобразувател от серийна комуникация „I²C“ към паралелна комуникация. По този начин микроконтролерът подава стойността за записване в клетка от паметта по I²C към преобразувателя, който „разширява“ серийния интерфейс към използванятия от паметта паралелен.

- Взаимодействие с тъчскрийн модула, който е свързан като периферно устройство към блок „Интерфейс към тъчскрийн дисплей“. Последният спомага за осъществяване на двустранна комуникацията между микроконтролера и отделните компоненти на тъчскрийн модула.

Цялото устройство, включващо както и генератора на произволни сигнали, така и развойната платка на течнокристалния дисплей, се захранва от две външни и несвързани помежду си захранвания - цифрово и аналогово. Те доставят нужните за генератора на сигнали цифрови и аналогови захранващи напрежения. Нуждата от две отделни захранвания се обосновава от наличието на цифрово-аналогов преобразувател (ЦАП) в електрическата верига, което е устройство съставено и от цифрова, и от аналогова схемотехника. Ако цифрово-аналоговият преобразувател се захрани от единен захранващ модул (т.е. цифровите и аналоговите захранващи напрежения се измерват спрямо един и същ общ проводник/земя), бързодействащата цифрова схемотехника ще внесе шумове⁽⁵⁾ в тихата аналогова схемотехника, които могат да се изразят в изкривявания на генерирания сигнал. Цифровото и аналоговото захранване се включват към блока „Интерфейс към захранването“ на генератора на сигнали.



фиг. 2.1: Блокова схема на цифров генератор на сигнали

ГЛАВА III

Проектиране на принципна електрическа схема на цифров генератор на сигнали

3.1 Елементна база и избор на основни интегрални схеми

Устройството е съставено от няколко интегрални схеми, взаимодействащи помежду си, както и допълнителните компоненти за правилното и оптимално им функциониране. В тази точка използваните интегрални схеми са описани и изборът им за прилагане в настоящата дипломна работа е обосновен.

3.1.1 Микроконтролер

DDS системата и тъчскрийн модулът са блоковете, които реализират както и желаната за един генератор на сигнали функционалност, така и потребителския интерфейс. Въпреки това те изискват друго устройство, което да ги управлява и настройва. В настоящата дипломна работа като свързващо звено между всички функционални системи на проектирания цифров генератор на сигнали е използван микроконтролер.

Използваният в устройството микроконтролер е ATmega2561. Последният е част от фамилията 8-битови микроконтролери megaAVR на Microchip. ATmega2561 е 64-пиново устройство с 256 KB флаш памет, 4 KB EEPROM памет, 8 KB SRAM памет, 54 входно-изходни извода с общо предназначение, 10-битов вграден АЦП, I²C интерфейс, SPI интерфейс, няколко 8-битови и няколко 16-битови таймери и много други вградени периферии. ATmega2561 е детайлно описан в каталожните му данни⁽⁷⁾.

Дипломантът е решил да използва микроконтролера ATmega2561, защото е запознат с megaAVR фамилията микроконтролери. От друга страна, управляващият софтуер, който реализира потребителския интерфейс на тъчскрийн дисплея, употребява сравнително голямо количество RAM памет. Понеже сложността на потребителския интерфейс не може да бъде определена, нужното количество вградена RAM памет на микроконтролера за реализиране на потребителския интерфейс също не може да бъде определено. Макар и да има по-евтини представители на фамилията megaAVR, с които би могла да се проектира успешно

принципната електрическа схема на генератора, потенциалната и вероятна нужда от голямо количество RAM памет обуславя избора на ATmega2561.

3.1.2 Статична памет с произволен достъп (SRAM)

Тази интегрална схема е една от основните съставни части на DDS системата. Изискванията към тази памет е да бъде високопроизводителна, понеже максималната честота на генерираия сигнал зависи от скоростта на достъп до клетките от паметта. Затова е избрана интегралната схема CY7C1021D от Cypress Semiconductor. Паметта е с много по-голям от нужния за проекта капацитет - съставена е от 65536 16-битови думи (1 Mbit), и съответно има 16 входа за адресация ($A_0 \div A_{15}$). Времето за достъп на интегралната схема е 10 ns, което означава, че паметта може да бъде обхождана с честота 100MHz. Друго удобство на този компонент е, че схемата може да се захрани с +5V и да работи с 5-волтови логически нива, с които работи микроконтролерът. Повече подробности за тази интегрална схема може да се намерят в каталожните ѝ данни⁽⁸⁾.

3.1.3 ЦАП

Понеже и този компонент е съставна част на DDS системата, се очаква, че избраният цифрово-аналогов преобразувател е качествен, високопроизводителен и шумоустойчив. Устройството е проектирано с възможността да се използва който и да е представител на семейството преобразуватели TxDAC на Analog Devices, понеже всички негови представители са съвместими помежду си.

За да се обоснови избора на TxDAC семейството за целите на дипломната работа, тук е разгледан конкретен представител на семейството - AD9764. Както всички ЦАП-ове, и това устройство включва както и аналогова, така и цифрова схемотехника. Понеже бързопревключващата цифрова част може да внесе нежелан шум в тихата аналогова част, тази интегрална схема има две отделни захранвания (цифрово - DVDD, и аналогово - AVDD), измервани спрямо две различни земи (съответно DCOM и ACOM). Въпреки че и цифровата, и аналоговата схемотехника на преобразувателя се захранват от постоянно напрежение със стойност +5V, за оптималната работа на преобразувателя захранващите му изводи не трябва да се свързват към единствен източник на захранващо напрежение. Скоростта на актуализация на изхода на този цифрово-аналогов преобразувател е 125 MSPS - достатъчно бързо за реализиране на генератора на сигнали. Други особености на преобразувателя са възможността за „заспиване“ (съществено намаляване на консумацията) на устройството,

взаимодействие с цифровата му част, използвайки 5-волтови логически нива, както и наличието на два комплементарни токови изхода с максимален сумарен изходен ток 20 mA. Повече подробности за AD9764 може да се открият в каталожните му данни⁽⁹⁾.

В принципната електрическа схема на проектирания генератор на сигнали се използва вече разгледаният AD9764. Понеже всички представители на TxDAC семейството са пиново-съвместими помежду си, AD9764 може да бъде заменен с други представители на същото семейство като AD9762⁽¹⁰⁾, AD9708⁽¹¹⁾ и др. Представителите на TxDAC серията преобразуватели се различават единствено по един свой параметър - резултацията, определяща минималната разлика в стойностите на две съседни подадени към преобразувателя входни числа. Разбира се, преобразувателите с по-висока резултация спомагат за генерирането на по-прецизен изходен сигнал, но обикновено са по-скъпи спрямо представителите на семейството с по-ниска резултация.

3.1.4 Двоични броячи и логически схеми

В блоковата схема от т. 2.2 (показана на фиг. 2.1) стана ясно, че обхождането на RAM паметта на DDS системата се осъществява чрез блока „Адресен брояч“. Последният трябва да бъде 16-битов двоичен, защото интегралната схема CY7C1021D има 16 пина за адресация ($A_0 \div A_{15}$).

За реализирането на 16-битов двоичен адресен брояч се използва фамилията от логически схеми 74HC. Тъй като тази фамилия няма интегрална схема, представляваща 16-битов двоичен брояч, блокът „Адресен брояч“ се реализира посредством два 12-битови двоични броячи 74HC4040⁽¹²⁾. Понеже последният няма извод за преносен бит, връзката между двета брояча се осъществява чрез допълнително навързани логически схеми от фамилията 74HC. Важна особеност на 74HC4040 е параметърът му „максимална честота“ (f_{max}), който има стойност 90MHz, когато интегралната схема е захранена от +5V и околната температура е равна на +25°C (стайна температура)⁽¹²⁾. Високата максимална честота (f_{max}) на броячите е необходима, защото те трябва да имат възможността да обхождат клетките на SRAM паметта на DDS системата изключително бързо.

3.1.5 Програмираме осцилатор

Програмируемият осцилатор е последната важна съставна част на DDS системата, като действието му се изразява с определяне скоростта на обхождане на SRAM паметта и съответно - честотата на генерирания сигнал. Следователно от този осцилатор се изисква

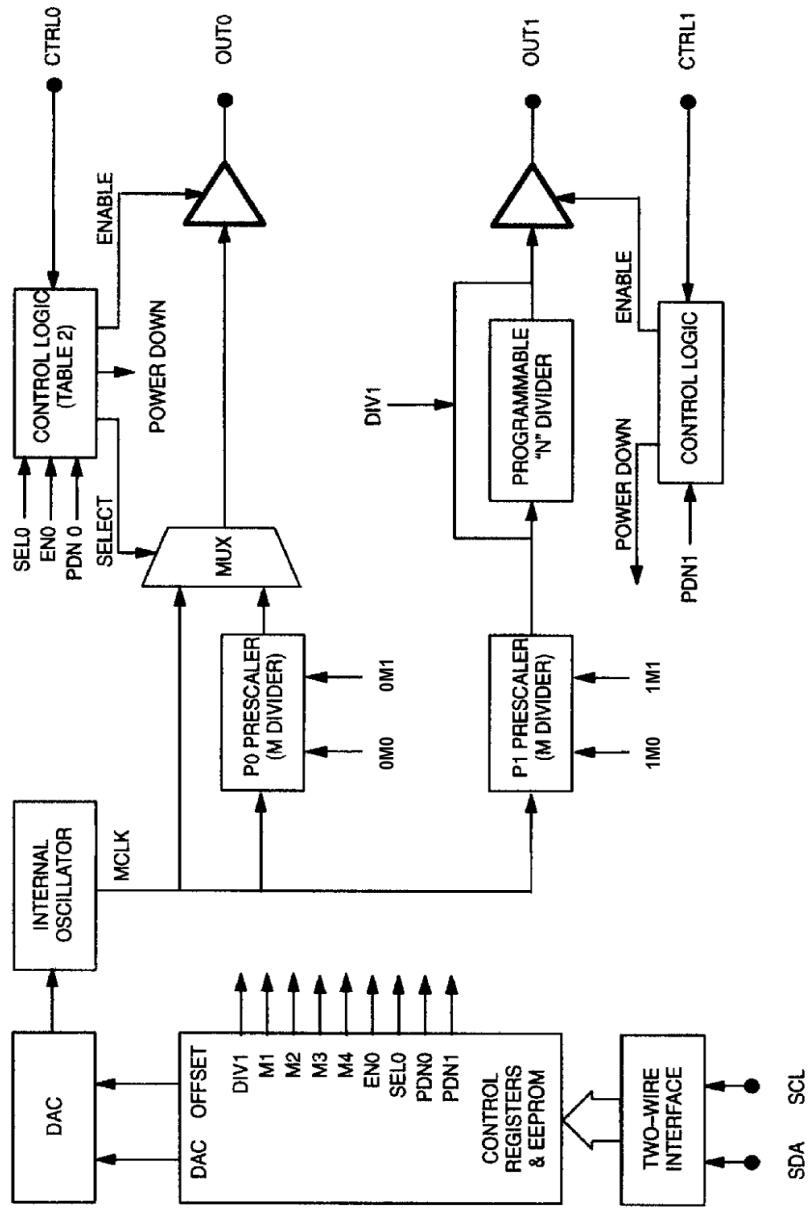
изключително прецизно регулиране на генерираната тактова честота в много голям честотен обхват.

В проектирания генератор на сигнали се използва интегралната схема DS1085-10. Основните предимства на този осцилатор са възможността за програмирането му чрез I²C интерфейс, захранването му от източник на напрежение със стойност +5V, два изхода - главен, който може да се регулира прецизно, и „референтен“, големия честотен обхват на главния му изход от 8.1kHz до 133MHz, както и голямата му прецизност, изразяваща се в ниската абсолютна грешка от 0.75%. DS1085-10 позволява регулирането на генерираната тактова честота през стъпки от 10kHz. Пълното описание на осцилатора е налично в каталожните му данни⁽¹³⁾.

На фиг. 3.1 е показана блоковата схема на програмируемия осцилатор DS1085, която е взета от каталожните му данни⁽¹³⁾. Устройството работи по следния начин: вграденият в интегралната схема осцилатор (означен на схемата като „INTERNAL OSCILLATOR“) генерира тактов сигнал между 66 MHz и 133 MHz, който се подава на двета изхода (главния и референтния). И двета изхода имат схеми тип делители (P0 и P1), които разделят тактовото трептене по 1, 2, 4 или 8. Обработеният от P0 тактов сигнал се подава директно на референтния изход **OUT0**. От друга страна, обработеният от P1 тактов сигнал може да се раздели допълнително от програмирам делител в обхвата от 2 до 1025, след което резултатното тактово трептене се подава на главния изход **OUT1**. Вграденият в устройството източник на трептения се настройва с помощта на регистър за отместване „OFFSET“ и 10-битов ЦАП. Преобразувателят се използва за прецизно инкрементално отместване на генерираната тактова честота в стъпки от 10 kHz. Но понеже един 10-битов ЦАП няма достатъчно голям брой стойности, който да позволи обхождането на всички възможни стойности в обхвата от 66MHz до 133MHz на стъпки от 10kHz, регистърът за отместване „OFFSET“ се използва за избиране на честотния обхват, който може да се обходи инкрементално на стъпки. Тези честотни обхвати са представени в таблица 6 от каталожните данни, която е дадена на фиг. A.1. Забележително е, че вграденият осцилатор може да се настрои да генерира честоти извън обхвата от 66MHz до 133MHz, но производителят не гарантира нормалното функциониране на устройството извън този обхват.

Осцилаторът има два допълнителни входа - **CTRL1** и **CTRL0**. И двета входни пина се програмират да изпълняват различни контролни функции, които са описани в таблица 3 и таблица 2 от каталожните данни на интегралната схема, които са дадени съответно на фиг. A.2 и на фиг. A.3.

Figure 1. DS1085 BLOCK DIAGRAM



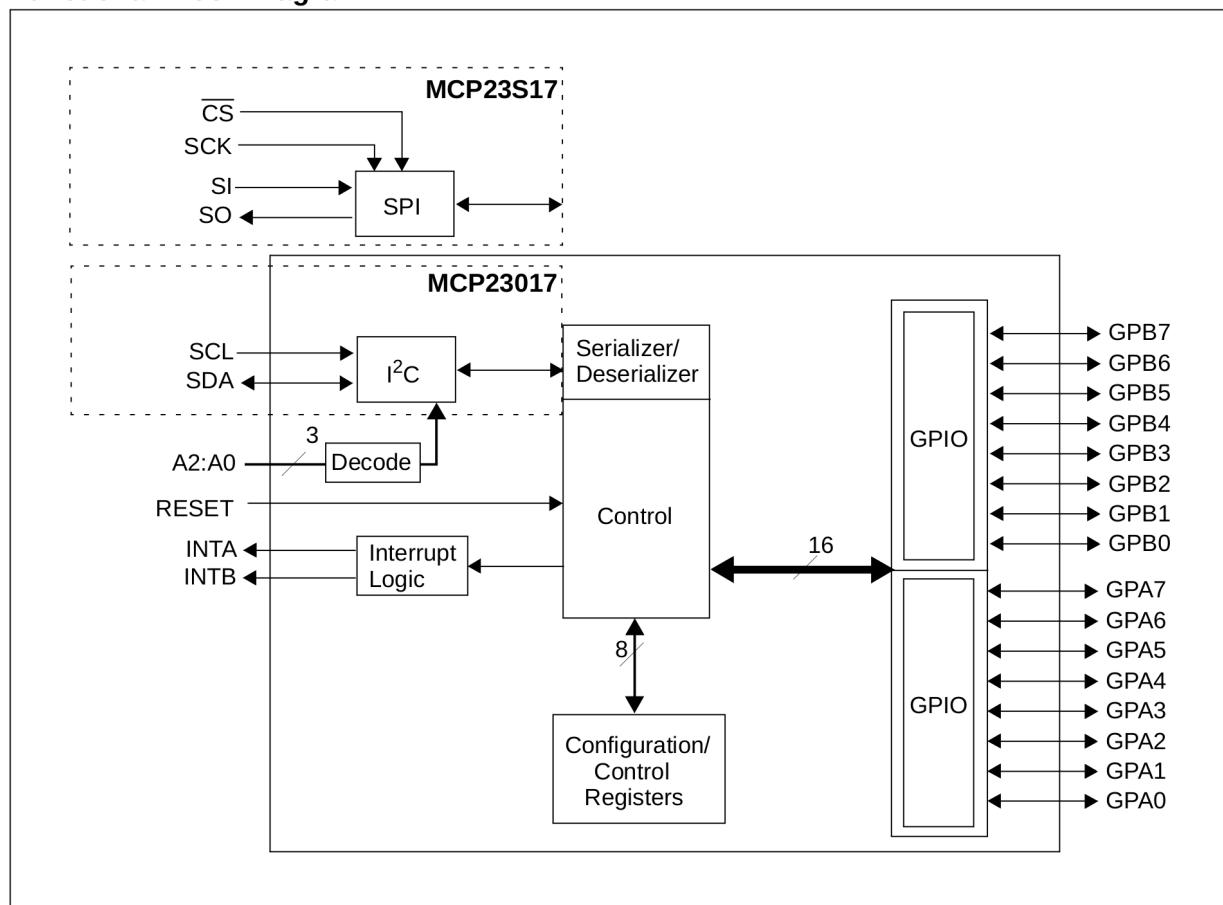
фиг. 3.1: Блокова схема на DS1085

3.1.6 Преходник последователна-паралелна комуникация

В проектирания генератор на сигнали се използва интегралната схема MCP23017 на Microchip, която представлява преходник от I²C интерфейс към 16-битов паралелен входно/изходен интерфейс. Устройството се използва при записване на данни от микроконтролера в SRAM паметта на DDS системата. Интегралната схема е описана подробно в каталожните ѝ данни⁽¹⁴⁾.

На фиг. 3.2 е представена блоковата схема, взета от каталожните данни, на MCP23017. Устройството се състои от два 8-битови входни/изходни порта с общо предназначение. Двата порта се настройват чрез конфигуриране на вътрешните за преходника регистри, които са дадени на фиг. А.4. Самите регистри се конфигурират посредством наличния на устройството I²C интерфейс.

Functional Block Diagram



фиг. 3.2: Блокова схема на MCP23017

3.1.7 Аналогови интегрални схеми

Използването на аналогова усилваща схемотехника в генератора на сигнали предоставя както и необходимото усилване по мощност на генерирания от DDS системата маломощен сигнал, така и възможността на потребителя да изменя амплитудата му и добавеното към него постояннотоковото отместване. За реализиране на блок „Изходна аналогова схемотехника“ се използват операционни усилватели AD8021 на Analog Devices. Тази интегрална схема се използва в проекта, защото представлява високоскоростен, малошумящ операционен усилвател, който може да се захрани с двойно захранване $\pm 12V$. Също така тези усилватели имат възможността да бъдат „изключени“, при което съществено се намалява консумираният от тях ток, чрез подаване на ниско логическо ниво на пин **DISABLE** спрямо пин **LOGIC REFERENCE**. AD8021 е описан детайлно в каталожните му данни⁽¹⁵⁾.

3.1.8 Цифров потенциометър

За добавяне на постояннотоково ниво, което да може да се изменя по цифров път (т.е. от микроконтролер) към генерирания сигнал, се използва интегралната схема MCP4551⁽³⁴⁾. Последната представлява цифров потенциометър, управляван с I²C интерфейс. Крайните изводи на цифровия потенциометър са **A** и **B**, а средният (пъзгача) - **W**. На фиг. 3.5 е показана вътрешната структура на потенциометъра. Общото съпротивление (R_{AB}) между крайните му изводи **A** и **B** е съставено от множество по-малки и равностойни съпротивления (R_S), чийто брой е 256. Чрез аналогов мултиплексор пъзгачът се свързва към точка, зададена от цифровия интерфейс, или между две от равностойните съпротивления (R_S), или между съпротивление и краен извод. Следователно двете съпротивления, образувани между пъзгача (**W**) и крайните изводи (**A** и **B**), се определят по следните формули:

$$R_{WA} = \frac{R_{AB} \times (256 - N)}{256} + R_W \quad [3.3]$$

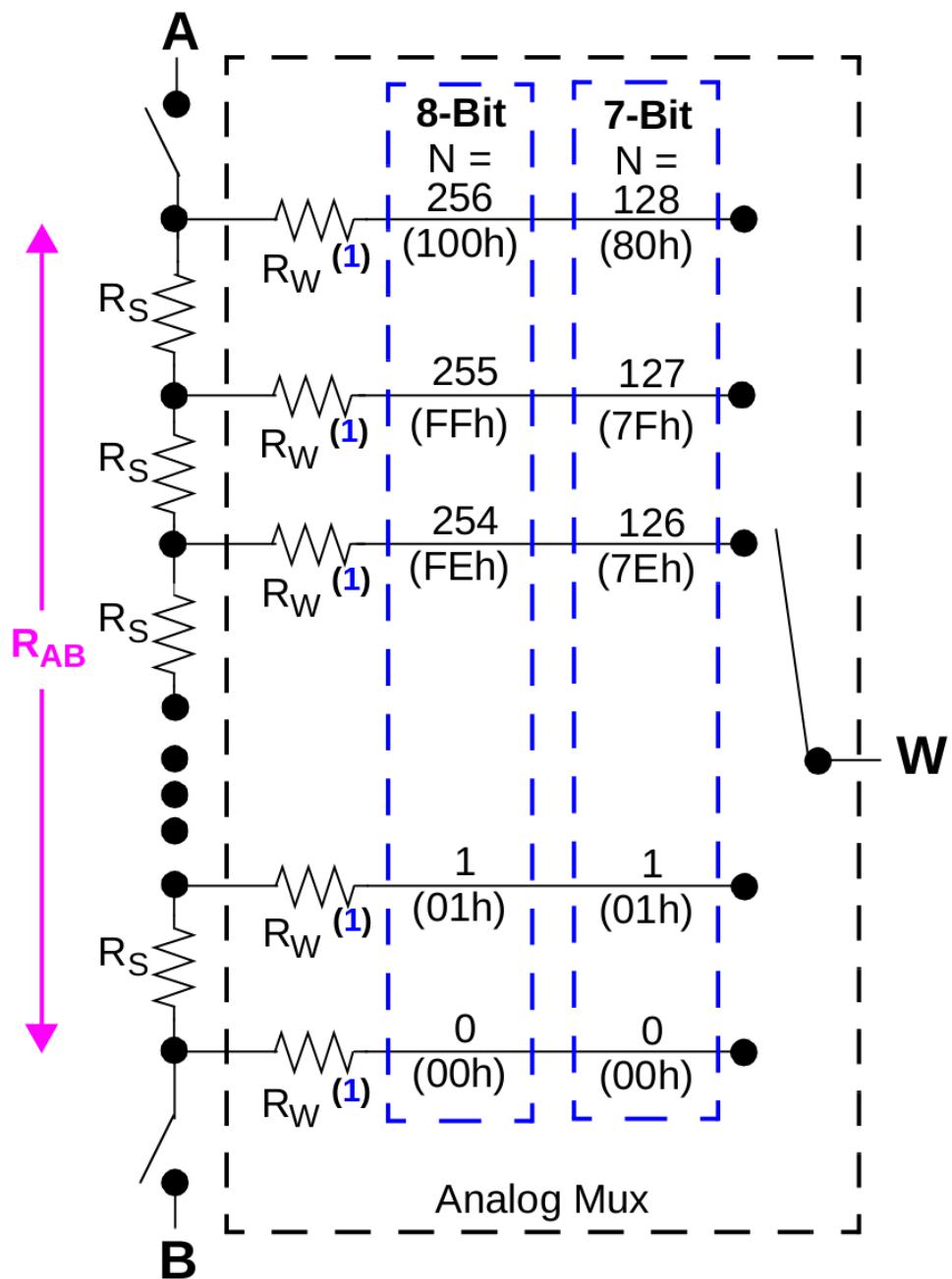
$$R_{WB} = \frac{R_{AB} \times N}{256} + R_W \quad [3.4]$$

където:

N - цифровата стойност, определяща към коя междинна точка да се включи пъзгача.

R_W - съпротивлението на пъзгача

Както се забелязва от формулите [3.3] и [3.4], плъзгачът (**W**) има малко съпротивление R_W , което е свързано последователно на съпротивленията между плъзгача и крайните изводи (**A** и **B**).



фиг. 3.5: Вътрешна структура на цифровия потенциометър MCP4551

3.2 Принципна електрическа схема

На фиг. 3.32, фиг. 3.33, фиг. 3.34, фиг. 3.35 и фиг. 3.36 е представена принципната електрическа схема, която е съставена от 5 страници, на проектирания в настоящата дипломна работа цифров генератор на сигнали. На фиг. 3.32 е показана главната страница на проекта, а останалите страници (от 2-ра до 5-та) представляват йерархични листа, включени като схемни блокове с входни и изходни сигнални линии. Йерархичните листа са представени в принципната електрическа схема по следния начин:

- Схемният лист/блок „DDS System“ на фиг. 3.32 включва втора страница (фиг. 3.33).
- Схемният лист/блок „16-bit Counter“ на фиг. 3.33 включва трета страница (фиг. 3.34).
- Схемният лист/блок „Output Circuitry“ на фиг. 3.32 включва четвърта страница (фиг. 3.35).
- Схемният лист/блок „TFT LCD Module Interface Block“ на фиг. 3.32 включва петата страница (фиг. 3.36).

Схемата е изработена от дипломанта, използвайки приложния програмен продукт „KiCad“.

3.2.1 Блок „Интерфейс към захранване“

Този блок включва всички компоненти, разположени в очертанието „POWER SUPPLY“ в горния десен ъгъл на 1-вия лист (фиг. 3.32). Предназначението на блока е да предостави всички необходими захранващи напрежения, използвани от генератора на сигнали.

Както е опоменато в т. 2.2, проектираното устройство се захранва от две отделни захранвания - цифрово и аналогово, които нямат обща връзка помежду си. Цифровите захранващи напрежения се отмерват спрямо цифровата земя (**DGND**), а аналоговите - спрямо аналоговата земя (**AGND**). Нужните за устройството захранващи напрежения са следните:

- +5V спрямо цифровата земя (**DGND**). Това постоянно напрежение се означава в схемата като „**+5VD**“.
- ±12V спрямо аналоговата земя (**AGND**). Тези постоянни напрежения се означават в схемата съответно като „**+12VA**“ и „**-12VA**“.

Двете захранвания се навързват към устройството посредством 2x5 пиновия съединител **J9**. Към първите четири пина (от 1-ви до 4-ти) се включва цифровото захранване, а към последните четири (от 7-ми до 10-ти) - аналоговото захранване. Пети и шести пин на J9

не са свързани. Получените три захранващи напрежения се изглеждат допълнително от кондензаторите **C10**, **C11** и **C12**, включени в очертанието „SMOOTHING CAPACITORS“.

Освен подаденото от цифровото захранване напрежение +5V устройството използва и цифровото захранващо напрежение +3.3V (спрямо **DGND**), означено в схемата като „+3.3V“. Линейният стабилизатор TC1017-3.3VCTTR⁽¹⁶⁾, означен на схемата като **U7**, приема на входа си (пин 1-ви) захранващото напрежение **+5VD** и изкарва желаното постояннотоково стабилизирано захранващо напрежение със стойност +3.3V на изхода си (пин 5-ти). Интегралната схема е от тип LDO линеен стабилизатор с $\pm 0.5\%$ точност и възможност за доставяне на 150mA постоянен ток. За оптималното функциониране на стабилизатора, производителя препоръчва добавянето на входен и изходен кондензатор съответно между входа (1-ви пин) и **DGND** и между изхода (5-ти пин) и **DGND**. На схемата (фиг. 3.32) кондензаторите **C14** и **C15** изпълняват функциите съответно на входен и изходен за стабилизатора кондензатор. **C14** и **C15** трябва да са керамични кондензатори от тип X7R или X5R с номинална стойност на капацитет 1uF. TC1017 може да бъде „изключен“ чрез подаване на ниско логическо ниво на входния пин **SHDN** (3-ти пин) спрямо **GND** (2-ри пин). Тъй като тази функционалност не е използвана в проектираното устройство, изводът **SHDN** е директно свързан към входа на стабилизатора (**V_{IN}**).

Също така генераторът на сигнали използва допълнително напрежение +5V спрямо аналоговата земя (**AGND**), означено на схемата като „+5VA“. Това напрежение се постига посредством линейния стабилизатор ADP7142AUJZ-5.0⁽¹⁷⁾, означен на схемата с **U6**. Този стабилизатор също е от тип LDO. Интегралната схема приема +12V на входа си (1-ви пин) и изкарва +5V на изхода си (5-ти пин) с точност $\pm 0.8\%$, като може да достави до 200mA постоянен ток (при оптимално осигурено охлажддане на печатната платка). За оптималното функциониране на линейния стабилизатор производителят препоръчва включването на стабилизиращи кондензатори към входа и към изхода на му. На схемата от фиг. 3.32 тези два кондензатора са означени съответно като **C13** и **C16**, които трябва да са керамични с диелектрик X7R или X5R и с номинален капацитет 2.2uF. ADP7142 може да бъде „изключван“, но понеже тази функционалност не се използва в генератора на сигнали, изводът **EN** (3-ти пин) е свързан директно към входа (**V_{IN}**) по препоръка на производителя. Изводът **SENSE/ADJ** е свързан към изхода на интегралната схема (**V_{out}**), следвайки образеца в каталожните данни⁽¹⁷⁾.

Флаговете, намиращи се в очертанието „POWER FLAGS“, се използват от KiCad и имат чисто документална функция.

3.2.2 Захранване на логически схеми и неизползвани логически елементи

В долния ляв ъгъл на 1-ви лист (фиг. 3.32) са разположени две съседни очертания - „UNUSED LOGIC GATES“ и „LOGIC IC POWER SUPPLIES“. Първото очертание включва логическите елементи, които са налични на съответните им интегрални схеми, но остават неизползвани в принципната електрическа схема. Другото очертание показва само захранващите пинове на използваните логически интегрални схеми от серията 74HC: два броя 74HC08⁽¹⁸⁾, един брой 74HC30⁽¹⁹⁾ и един брой 74HC04⁽²⁰⁾. Всяка схема е захранена с цифровото напрежение +5V_D. Между захранващите им изводи (V_{CC}) и цифровата земя (GND) са включени т. нар. отделящи кондензатори (от англ.: „decoupling/bypass capacitors“). Тяхната функция е да шунтират всякакви променливотокови шумове, които може да се появят на постояннотоковите захранващи линии и може да попречат на нормалното функциониране на интегралните схеми. Широко прието е в практиката между захранващите изводи на всяка интегрална схема и общия проводник (земята) да се добави керамичен отделящ кондензатор с номинален капацитет 100nF. На схемата **C3**, **C4**, **C5** и **C8** са отделящи кондензатори, свързани съответно към **U2**, **U3**, **U4** и **U1**.

3.2.3 Блок „Микроконтролер и допълнителна периферия“

Този блок включва всички компоненти в очертанието „MICROCONTROLLER“ на фиг. 3.32.

Използваният микроконтролер - ATmega2561, означен на схемата с **U5**, е захранен от цифровото напрежение +5V_D. Изводът V_{CC} е цифровото захранване на микроконтролера, докато изводът AV_{CC} е свързан към захранването на PORT F и вграденото му АЦП. AV_{CC} е свързан към V_{CC} по препоръка на производителя от каталожните данни на устройството. **C6**, **C7** и **C47** са отделящи кондензатори, свързани съответно към пин 21-ви (V_{CC}), пин 64-ти (AV_{CC}) и пин 52-ри (V_{CC}), който не е изображен на схемния символ на ATmega2561.

Към извода AREF на **U5** е свързано аналоговото опорно напрежение, което е необходимо за функционирането на вграденото АЦП. Опорното напрежението може да бъде или вътрешно генерирано от интегралната схема, или външно свързано към извода AREF. И в двата случая производителят препоръчва включването на отделящ кондензатор, означен на принципната електрическа схема с **C9**, към AREF за получаване на изчистено от шумове опорно напрежение (стр. 269, т. 26.2 от (7)). Съединителят **J8** е свързан към AREF, доставяйки възможност на потребителя да свърже външно генерирано опорно напрежение.

За изпълняването на програмни инструкции микроконтролерът се нуждае от източник на тактови трептения. Затова към изводите **XTAL1** и **XTAL2** е свързан кварцов осцилатор (стр.41, фиг. 10-2 от (7)) с честота на трептене 16MHz. Между изводите на осцилатора **Y1** и **GNDD** са свързани два керамични кондензатора **C1** и **C2** с номинален капацитет 22pF. **Резисторът R1 е погрешно поставен на схемата от дипломанта и трябва да се пренебрегне.**

Входният извод **RESET** се използва за рестартиране на ATmega2561, при което микроконтролерът започва да изпълнява записаната в паметта му програма от началото. За изпълняване на тази операция е необходимо да се подаде ниско логическо ниво на **RESET**. Потребителят има възможността ръчно да рестартира **U5**, натискайки бутона **SW1**. **R2** и **SW1** образуват делител на напрежение и са свързани в схема тип „pull-up“, при която на средната им точка, свързана към **RESET**, в нормално състояние (ненатиснат бутон) се отделя високо логическо ниво (+5V), защото **SW1** представлява отворена верига и може да се представи като безкрайно голямо съпротивление. При натискане на бутона **SW1** линията **RESET** се свързва накъсо към земята (**GNDD**), при което извода **RESET** на микроконтролера получава ниско логическо ниво и той се рестартира. Паралелно на **R2** е включен диодът **D1**, който защитава интегралната схема **U5** от електростатични разряди⁽²¹⁾.

ATmega2561 разполага с шест 8-битови порта (**PORT A**, **PORT B**, **PORT C**, **PORT D**, **PORT E**, **PORT F**) и един 6-битов (**PORT G**).

Към първите четири пина на **PORT B** (**PB0** ÷ **PB3**), означени съответно с табелите **SS**, **SCK**, **MOSI** и **MISO**, е вътрешно свързан вграденият хардуерно-реализиран SPI интерфейс на микроконтролера. Този интерфейс е изведен на съединителя **J2**. От друга страна SPI интерфейсът се използва за програмиране на микроконтролера чрез 6-пиновия конектор **J1**. Серийният интерфейс се използва и за осъществяване на комуникация между микроконтролера и отделните части на тъчскрийн модула. Останалите четири пина (**PB4** ÷ **PB7**) се използват за подаване на различни контролни сигнали на съставящите части на генератора на сигнали:

- **PB4 - CMR1**: нулира стойността на първия 12-битов брояч (**U13** от фиг. 3.34), подавайки високо логическо ниво.
- **PB5 - CMR2**: нулира стойността на втория 12-битов брояч (**U14** от фиг. 3.34), подавайки високо логическо ниво.
- **PB6 - R/W**: избира режима, в който работи SRAM паметта (**U9** от фиг. 3.33) на DDS системата. При подаване на високо логическо ниво паметта работи в режим на четене, при подаване на ниско ниво - режим на писане.

- **PB7 - CP**: инкрементира стойността на 16-битовия адресен брояч при подаване на ниско логическо ниво.

Към първите два пина на **PORT D (PD0 и PD1)**, означени съответно с табелите **SCL** и **SDA**, е вътрешно свързан вграденият хардуерно-реализиран I²C интерфейс на микроконтролера. Останалите шест пина (**PD2 ÷ PD7**) се използват за осъществяване на комуникация с тъчскрийн модула, изобразен на 5-та страница (фиг. 3.36):

Изводите **PF0**, **PF1** и **PF2** също се ползват като контролни сигнали за DDS системата:

- **PF0 - PWRDWN**: „изключва“ интегралните схеми, съставящи DDS системата, при което съществено се намалява консумацията на генератора на сигнали.
- **PF1 - IOExRESET**: рестартира преходника от I²C към паралелна комуникация (**U10** от фиг. 3.33), подавайки ниско логическо ниво.
- **PF2 - OSCOE**: изключва изхода на програмируемия генератор на тактови трептения на DDS системата (**U8** от фиг. 3.33).

За правилното функциониране на I²C комуникационния интерфейс е наложително да се добавят т. нар. pull-up резистори към проводящите проводници **SCL** и **SDA**, понеже в състояние на покой (при което не протича комуникация) и двете линии трябва да са във високо логическо ниво (+5V). На схемата от фиг. 3.32 тази функция се извързва от резисторите **R3** и **R4**. Те са оразмерени по указанията, описани в (22). Тук са дадени само извършените изчисления, използвайки скоростта на комуникация I²C Fast Mode (400kHz):

$$R_p(\max) = \frac{t_r}{(0.8473 * C_b)} = \frac{300 * 10^{-9}}{(0.8473 * 50 * 10^{-12})} \approx 7.08 k\Omega \quad [3.6]$$

$$R_p(\min) = \frac{V_{CC} - V_{OL}(\max)}{I_{OL}} = \frac{5 - 0.4}{(3 * 10^{-3})} \approx 1.53 k\Omega \quad [3.7]$$

където:

$$C_b = 50 pF,$$

$$V_{CC} = +5V$$

Важно е да се поясни, че сумарният паразитен капацитет на I²C магистралата C_b е 50pF, защото към нея са навързани пет I²C устройства, като всяко едно може да има максимален паразитен капацитет от 10pF, както е посочено в таблица 9 (стр. 47 от (23)) от официалната I²C спецификация⁽²³⁾.

Според [3.6] и [3.7] за номинално съпротивлението на **R3** и **R4** е избрана стойността 2kΩ.

Изводът **PF3** е означен с табелата **OCAL** и е свързан към 3-тия канал (**ADC3**) на вграденото АЦП на ATmega2561. Каналът се използва за настройване на амплитудата на генеририания сигнали (виж т. 3.2.6).

Изводът **PG5**, означен с табелата **BL_LCD**, е свързан към PWM изход B на таймер 0 (**OC0B**) на ATmega2561. Този пин се използва за настройване на яркостта на течнокристалния еcran (виж т. 3.2.4).

Останалите пинове - **PA0 ÷ PA7, PC0 ÷ PC7, PE0 ÷ PE7, PF4 ÷ PA7** и **PG0 ÷ PG4**, на ATmega2561 не се използват в проекта. Те са изведени съответно към пиновите съединители **J6, J7, J5, J4** и **J3**.

3.2.4 Блок „Интерфейс към тъчскрийн модула“

Този блок включва всички елементи, разположени на 5-тия лист (фиг. 3.36) на принципната електрическа схема. Целта на блока е да предостави интерфейс за комуникация между микроконтролера и тъчскрийн модула.

На фиг. 3.36 тъчскрийн модулът е представен чрез схемния символ **DS1**, който има само документална функция, т.е. не е разположен върху печатната платка. Пиновете от 1-ви до 9-ти се използват за комуникация с течнокристалния еcran (т.е. с драйверната интегрална схема ILI9341). Модулът трябва да се свърже към платката на цифровия генератор посредством 9-пиновия съединител **J13**. Пиновете от 10-ти до 14-ти са свързани към тъчскрийн контролера (XPT2046) на развойната платка и трябва да се свържат към платката на генератора чрез 5-пиновия конектор **J12**. Пиновете от 15-ти до 18-ти на **DS1** са свързани към 4-пиновия съединител **J11** и се използват за комуникация с SD картата в гнездото на тъчскрийн модула.

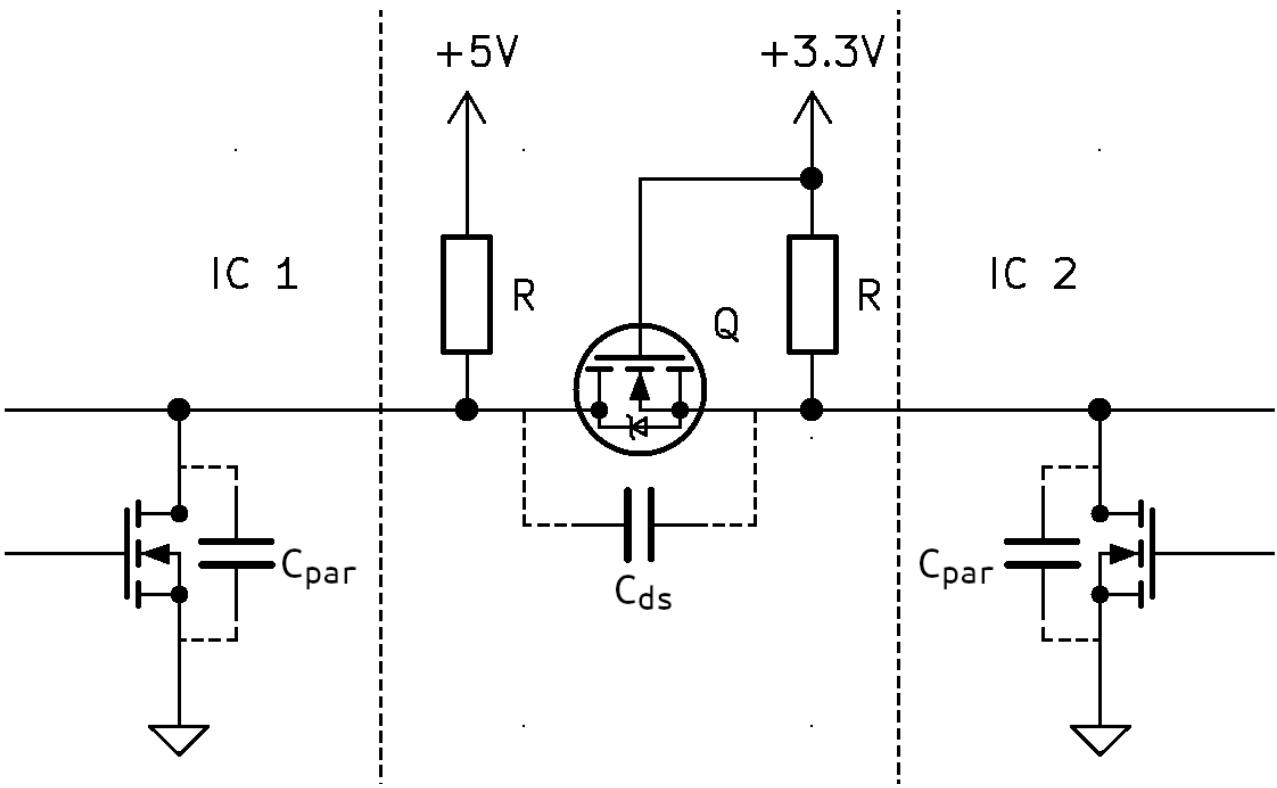
На фиг. A.5 е дадена схема с периферните за развойната платка на модула елементи. Анализирайки схемата, става ясно, че компонентите на развойната платка се захранват от линийния стабилизатор **U1**, който изкарва напрежение +3.3V на изхода си. На входа на същия стабилизатор се подава захранващото напрежение +5VD (спрямо **GNDD**) от платката на цифровия генератор чрез съединителя **J13**.

Цифровите интегрални схеми, използвани в цифровия генератор на сигнали, работят с +5V логически нива. От друга страна, компонентите на развойната платка на тъчскрийн модула функционират с +3.3V логически цифрови нива. Поради несъответствието в използваните логически нива се налага да се употребят схеми тип преобразуватели на логическо ниво. MOS транзисторите **Q4 ÷ Q12** на фиг. 3.36, както и свързаните към тях

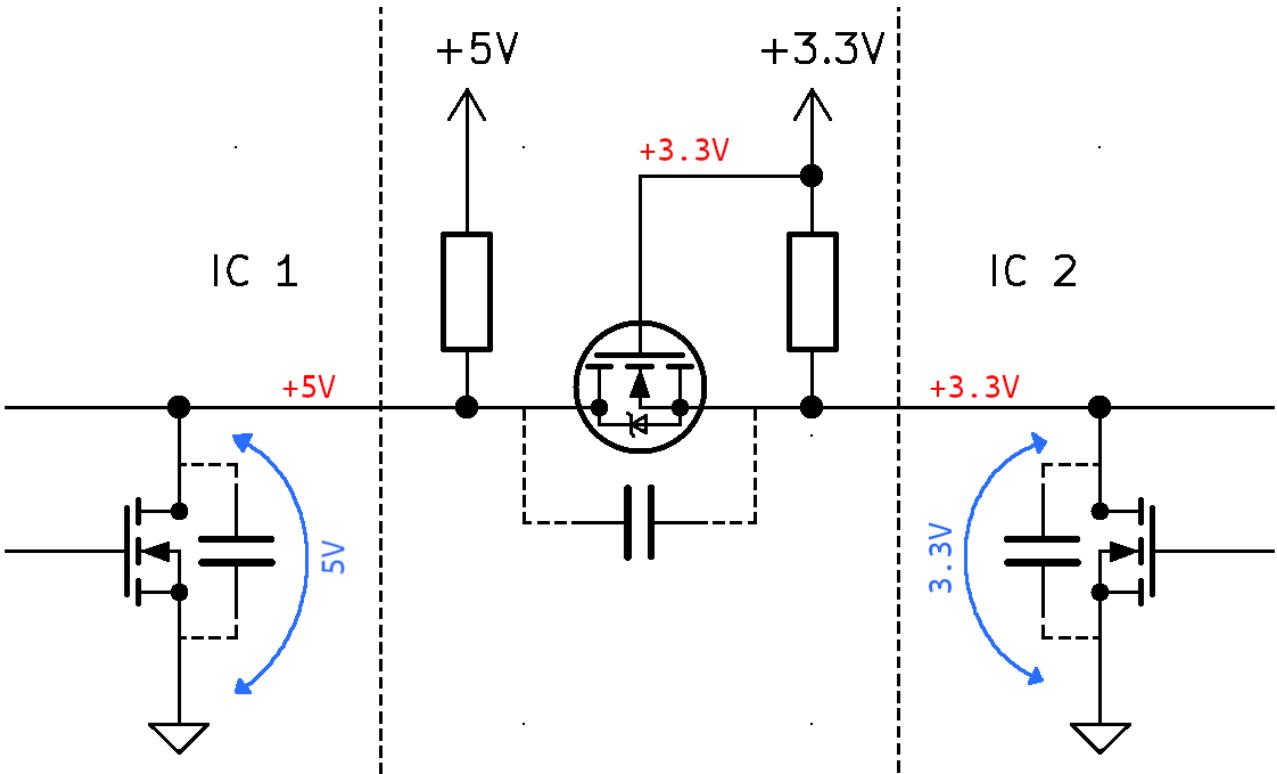
резистори, представляват преобразуватели на логически ниво от +5V на +3.3V. Принципна схема на един такъв преобразувател е представена на фиг. 3.8. Схемата се състои от един NMOS транзистор (означен с Q) и два резистора с еднакво съпротивление (означени с R). Преобразувателят е свързан между два входно-изходни извода, представени чрез интегрални NMOS транзистори, на две различни интегрални схеми.

В нормално състояние по линията между двата входно-изходни извода се предава високо логическо ниво. На дрейна на Q е наличен потенциал от +5V, а на сорса - +3.3V. Понеже гейта на Q е свързан директно към захранващото напрежение +3.3V, следва, че напрежение гейт-сорс е нулево, а напрежението гейт-дрейн - отрицателно. Следователно NMOS транзисторът Q е запущен. Това състояние на проводящата линия е представено графично на фиг. 3.9.

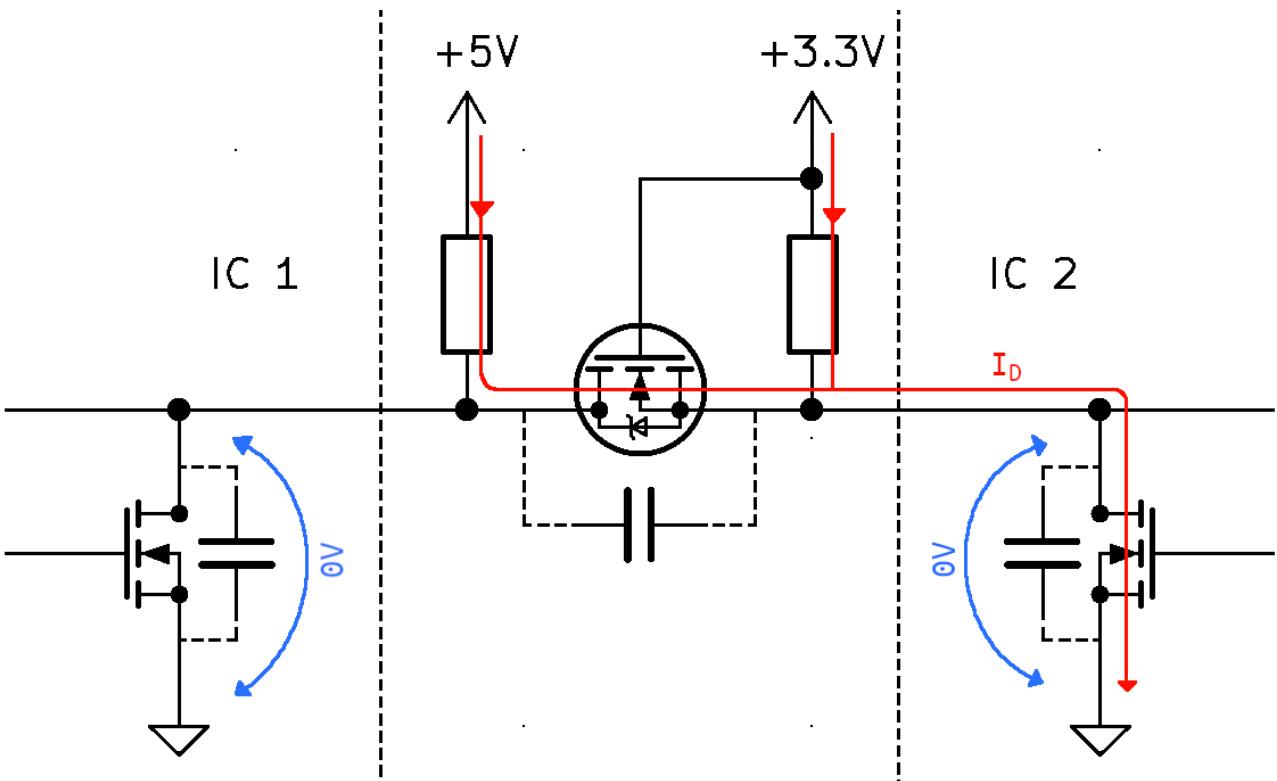
При предаване на логически ниско ниво от страна на по-ниското захранващо напрежение (+3.3V) електрическият потенциал на сорса на Q е 0V. Понеже гейта е свързан директно към +3.3V, напрежението гейт-сорс е 3.3V и следователно транзисторът се отпушва. Понеже съпротивлението дрейн-сорс на MOS транзисторите в отпушено състояние ($R_{DS(ON)}$) типично е минимално (в областта на $m\Omega$) потенциалите на дрейна и сорса на Q практически се изравняват, т.е. към комуникиращата страна с +5V се предава логическа „0“. Важно е да се отбележи, че при отпушване на транзистора протичат два постоянни токове между резисторите и входно-изходния извод на комуникиращата страна с +3.3V (по-ниското захранващо напрежение). Следователно при предаване на логическа „0“ схемата има консумация по постоянен ток. Предаването на логическа „0“ от страната с +5V (по-голямото захранващо напрежение) е аналогично, понеже силициевата структура на MOS транзисторите е симетрична и следователно транзисторът Q работи като електронен ключ и в двете посоки. Предаване на логическа „0“ през преобразувателя е представено графично на фиг. 3.10.



фиг. 3.8: Принципна електрическа схема на преобразувател на логическо ниво



фиг. 3.9: Предаване на логическа „1“ през схемата преобразувател



фиг. 3.10: Предаване на логическа „0“ през схемата преобразувател

Особеност на схемата от фиг. 3.8 е наличието както и на паразитния капацитет дрейн-сорс (C_{ds}) на транзистора Q , така и паразитните капацитети (C_{par}) на изводите на комуникиращите интегрални схеми. Тези паразитни капацитети трябва да са минимални, понеже образуват в комбинация с резисторите образуват нискочестотен филтър. Честотата на отсичане (f_{-3dB}) на филтъра се влияе от съпротивлението на резисторите R и от сумарния паразитен капацитет по проводящата линия и се изчислява по [3.11]. От една страна, намаляването на съпротивленията R от фиг. 3.8 повишават честотата на отсичане, но за сметка на това увеличават постояннотоковата консумация, изчислена по [3.12], на схемата. Следователно необходимо е схемата да се проектира с достатъчно голяма честота на отсичане, която да не оказва влияние на комуникацията на схемата, и с минимална постояннотокова консумация.

$$f_{-3dB} = \frac{1}{2\pi R C_{bus}} \quad [3.11]$$

$$I_D = \frac{V_{CC(H)}}{R} + \frac{V_{CC(L)}}{R} \quad [3.12]$$

където:

C_{bus} - сумарният паразитен капацитет на проводящата линия

$V_{CC(H)}$ - по-високото захранващо напрежение

$V_{CC(L)}$ - по-ниското захранващо напрежение

В преобразувателите на логическо ниво от фиг. 3.36 се използват резистори с $\pm 5\%$ толеранс. Колкото по-голяма е стойността на реалното съпротивление спрямо номиналното за резистора, толкова повече се намалява и консумираният ток, и честотата на отсичане (f_{-3dB}). В обратния случай (когато реалното съпротивление е по-малко от номиналното) и консумираният ток, и честотата на отсичане (f_{-3dB}) се увеличават.

MOS транзисторите, използвани в преобразувателите са AP2310GN, които имат паразитен капацитет дрейн-сорс (C_{ds}), равен на 20pF , изчислен с израза [3.13]. Микроконтролерът представлява комуникиращата с по-високо напрежение ($+5\text{V}$) страна и има максимален паразитен капацитет на всеки пин 10pF (таблица 31-7 от (7)). Максималните паразитни капацитети на SD карта, тъчскрийн контролера и LCD контролера, представляващи комуникиращата с по-ниското напрежение ($+3.3\text{V}$) страна, са съответно 10pF (таблица 6-6 от (27)), 15pF (стр. 6 от (26)) и $30\text{pF}^{(25)}$.

$$C_{ds} = C_{oss} - C_{rss} = 70\text{ pF} - 50\text{ pF} = 20\text{ pF} \quad [3.13]$$

Преобразувателите от фиг. 3.36 са проектирани с цел постигане на серийна комуникация с SD картата, LCD драйвера (ILI9341) и тъчскрийн контролера (XPT2046) съответно със скорости 4MHz , 4MHz и 2MHz . В таблица 3.14 са представени оптималните резултати от оразмеряването на преобразувателите, вземайки под внимание изразите [3.11] и [3.12], както и всички изброени досега параметри, които оказват влияние на изразите.

ПАРАМЕТЪР	БЕЛЕЖКА	УСТРОЙСТВО		
		ILI9341 4 MHz	XPT2046 2 MHz	SD card 4 MHz
R_{HOM} , Ω	-	560	1600	910
R_{max} , Ω	при +5%	588	1680	955.5
R_{min} , Ω	при -5%	532	1520	864.5
C_{ds} , pF	AP2310GN	20	20	20
C_{IC1} , pF	AVR μ C	10	10	10
C_{IC2} , pF	УСТРОЙСТВО	30	15	10
C_{bus} , pF	Сумарен Капацитет	60	45	40
$f_{-3\text{dB}(\text{min})}$, MHz	при R_{max}	4.51	2.11	4.16
$I_{\text{D}(\text{max})}$, mA	при R_{min}	15.6	5.5	9.6

таблица 3.14: Оптимални резултати от оразмеряване на преобразувателите на логическо ниво

Тъй като и трите устройства (XPT2046, SD card, ILI9341), комуникиращи с +3.3V логически нива, се управляват от един и същ SPI порт на микроконтролера ATmega2561, преобразувателите на SPI линиите (**MISO**, **MOSI**, **SCK**) са оразмерени с най-ниското съпротивление, задоволяващо комуникационните нужди на всяко устройство. В този случай това съпротивление е с номинална стойност 560 Ω .

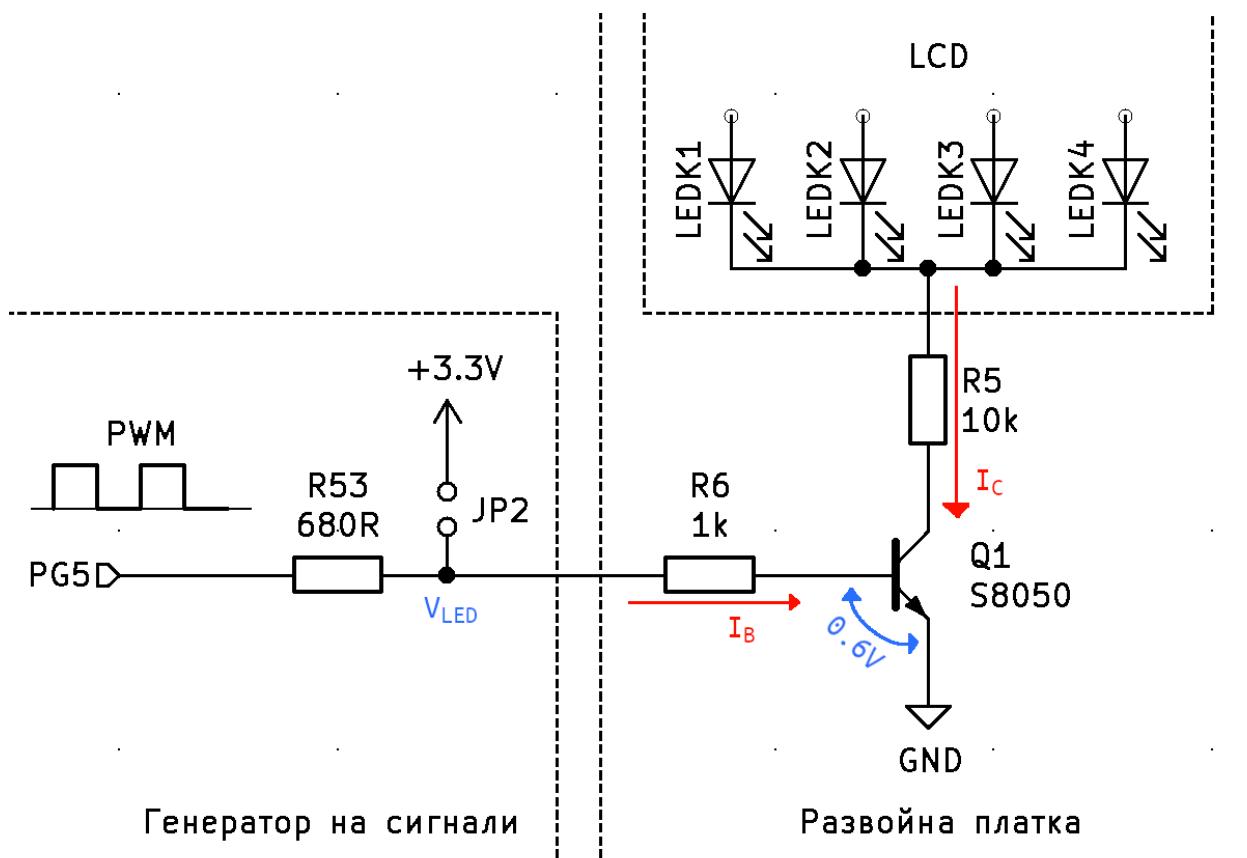
На фиг. 3.36 се вижда, че **LED** изводът (8-ми) на тъчскрийн модула, който е означен с табелата **LCD_BL**, е свързан от една страна към захранващото напрежение +3.3V чрез джъмпера **JP2**, а от друга - към **PG5** на микроконтролера посредством резистора **R53**. Изводът **LED** се използва за настройване на яркостта на течнокристалния экран. Разглеждайки схемата на фиг. A.5, става ясно, че яркостта се контролира от биполярния транзистор **Q1**, свързан с колектора си към катодите на светодиодите, съставящи экрана. Принципната схема на описаното свързване е представена на фиг. 3.18. Транзисторът контролира яркостта на экрана чрез изменяне на колекторния си ток (I_C), протичащ през светодиодите на экрана. Колекторният ток (I_C) на биполярен транзистор е функция на базовия ток (I_B), както е показано на [3.15]. Следователно чрез изменение на базовия ток (I_B) на **Q1** е възможно да се контролира яркостта на экрана. Изменението на базовия ток се постига чрез Широчинно-Импулсна Модулация (PWM), генерирана от микроконтролера ATmega2561 чрез пин **PG5**. Резисторът **R53** е поставен между пин **PG5** и **LED** (8-ми) извода на тъчскрийн модула, защото **LED** изводът очаква напрежение до +3.3V, а микроконтролерът работи с +5V логически нива. Изборът на стойността на съпротивлението **R53** се обуславя от изразите

[3.16] и [3.17]. В израз [3.16] е изчислен базовият ток (I_B), който се очаква при подаване на напрежение +3.3V на LED извода на тъчскрийн модула. В израз [3.17] е изчислена оптималната стойност на R53 за постигане на същия базов ток при подаване на високо ниво (+5V) от пин PG5 на микроконтролера. Номиналната стойност 680Ω е избрана, понеже е най-близка до изчисленото оптимално съпротивление от [3.17]. В случай, че потребителят желае постоянна максимална яркост, която не се контролира от микроконтролера, може да постави джъмпера JP2.

$$I_C = \beta I_B \quad [3.15]$$

$$I_B = \frac{V_{LED} - V_{BE}}{R6} = \frac{3.3V - 0.6V}{1k\Omega} = 2.7mA \quad [3.16]$$

$$R53 = \frac{V_{PG5} - V_{LED}}{I_B} = \frac{5V - 3.3V}{2.7mA} \approx 630\Omega \quad [3.17]$$



фиг. 3.18: Управление на яркостта на течнокристалния екран

3.2.5 Блок „DDS система“

Този блок обхваща цялата DDS система, която е разположена на 2-ри лист (фиг. 3.33) на принципната електрическа схема на генератора. Блокът е разделен на пет очертания, които са съставните части на системата.

3.2.5.1 Очертание „SRAM памет“

В средата на схемата е разположена интегралната схема CY7C1021D-10ZSXI - SRAM паметта на възпроизвеждащата система. Тя е означена на схемата с **U9**. Интегралната схема има два захранващи извода (11-ти и 33-ти), които са захранени от **+5VD**, и два заземяващи извода (12-ти и 34-ти), свързани към цифровата земя (**GNDD**). Към захранващите пинове - 11-ти и 33-ти, са свързани съответно и отделящите кондензатори **C18** и **C20**. Паметта има пет извода, чрез които се контролира поведението ѝ:

- **CE** (Chip Enable): „изключва“ интегралната схема при подаване на високо логическо ниво.
- **WE** (Write Enable): позволява записването на клетка от паметта. Активен при ниско логическо ниво.
- **OE** (Output Enable): позволява четенето на стойностите, записани в клетките на паметта. Активен при ниско логическо ниво.
- **BHE** (Byte High Enable): Позволява четенето и писането във високия байт на клетка памет (**I/O₈ ÷ I/O₁₅**). Активен при ниско логическо ниво. Изводите **I/O₈ ÷ I/O₁₅** заемат състояние на висок импеданс в противен случай.
- **BLE** (Byte Low Enable): Позволява четенето и писането в ниския байт на клетка памет (**I/O₀ ÷ I/O₇**). Активен при ниско логическо ниво. Изводите **I/O₀ ÷ I/O₇** заемат състояние на висок импеданс в противен случай.

Изводът **CE** е свързан към **PWRDWN** сигналната линия, управлявана от микроконтролера. По този начин при „изключване“ на генератора, т.е. подаване на високо ниво на **PWRDWN**, SRAM паметта се изключва. В противен случай интегралната схема е активна.

Тъй като всички 16 бита на клетките от паметта се използват при генериране на сигнали, изводите **BLE** и **BHE** са заземени (към **GNDD**). По този начин и високият, и ниският байт са винаги активни.

Разглеждайки таблицата на истинност на CY7C1021D, която е дадена на фиг. A.6, става ясно, че в режим на писане в паметта сигналът **WE** е в ниско ниво, а състоянието на

сигнала \overline{OE} е без значение. От друга страна, в режим на писане е необходимо сигналът \overline{WE} да е във високо ниво, а \overline{OE} - в ниско. Описаното поведение предоставя възможност за контролиране на SRAM паметта само с един контролен сигнал, обозначен като R/\overline{W} , който е контролиран от микроконтролера. Изводът \overline{WE} е свързан директно към проводящата линия R/\overline{W} , докато изводът \overline{OE} е свързан към нея през един от инверторните логически елементи на **U1** (74HC04). Описаният начин на свързване гарантира, че контролните сигнали \overline{WE} и \overline{OE} винаги са в противоположни логически състояния. При подаване на високо ниво на линията R/\overline{W} сигналът \overline{OE} приема ниско ниво, а сигналът \overline{WE} - високо, при което SRAM паметта е в режим на четене. При подаване на ниско ниво на линията R/\overline{W} се обръщат състоянията на контролните сигнали на паметта и тя е в режим на писане.

Изводите $A_0 \div A_{15}$ са адресните пинове на паметта, чрез които се адресира специфична нейна клетка. Те са свързани съответно към изходите на адресния брояч ($Q_0 \div Q_{15}$).

Изводите $I/O_0 \div I/O_{15}$ са входно-изходните пинове на паметта. Те се използват както и за подаване на стойността на клетките към цифрово-аналоговия преобразувател на системата, така и за записване на стойността на клетките от MCP23017.

3.2.5.2 Блок „16-битов адресен брояч“

Адресният брояч се използва за адресиране на SRAM паметта на DDS системата. Той е представен символично на 2-ра страница (фиг. 3.33). Има 16 изхода ($Q_0 \div Q_{15}$) които са свързани към адресните пинове ($A_0 \div A_{15}$) на **U9**. Освен това поведението на адресния брояч се контролира с 3 контролни сигнала - **MR1**, **MR2** и \overline{CP} . Самият адрессен брояч е съставен от няколко логически интегрални схеми, като структурата му е представена на 3-тия лист (фиг. 3.34) от принципната електрическа схема.

Адресният брояч е съставен от два 12-битови двоични брояча 74HC4040⁽²⁴⁾. И двета брояча са захранени от **+5VD** и заземени към **GNDD**, като към захранващите им изводи (**VCC**) са включени отделящите кондензатори **C28** и **C29**. Изходите $Q_0 \div Q_{11}$ на първия брояч, означен с **U13**, представляват 12-те младши изхода ($Q_0 \div Q_{11}$) на адресния брояч, а изходите $Q_0 \div Q_3$ на втория брояч са 4-те старши изхода ($Q_{12} \div Q_{15}$) на адресния брояч.

Изводът **MR** (Master Reset) (11-ти пин) на двоичния брояч 74HC4040 нулира стойността му. Двата пина за нулиране на **U13** и **U14** са свързани съответно към контролните сигнали **CMR1** и **CMR2**, които се управляват от микроконтролера.

Стойността на броячите се инкрементира (увеличава с единица) като се подаде ниско ниво на техните изводи \overline{CP} . Входният пин \overline{CP} на първия брояч (**U13**) е свързан към пин **PB7**.

на ATmega2561, позволящ на микроконтролера да задава стойността на адресния брояч без намесата на програмируемия осцилатор.

Понеже двоичният брояч 74HC4040 няма извод с преносен бит, **U13** и **U14** са свързани помежду си посредством допълнителни логически интегрални схеми от серията 74HC. Изходите ($Q_0 \div Q_{11}$) на първия брояч се свързват към логически елементи, съставящи интегралните схеми **U2** и **U3**, от типа двувходови „логическо И“ (AND). Резултатите от логическото умножение са означени с табелите **CP2I0** \div **CP2I5** и се подават на входовете на логическата схема **U4**, която е от тип осемвходово „логическо НЕ-И“. С табелата **CP2I6** се означава инвертирианият входен сигнали \overline{CP} , който се свързва към един от входовете на **U4**. Последният вход на **U4** е свързан директно към **+5VD**, т.е. към входа винаги е подадено високо логическо ниво. По този начин след нулиране на броячите, изходът на **U4**, който е свързан към входния пин \overline{CP} на **U14**, се намира във високо логическо ниво. Когато първият брояч (**U13**) достигне своята максимална стойност (изходите $Q_0 \div Q_{11}$ са активни) и постъпи сигнал за увеличаване стойността на адресния брояч (\overline{CP}), осемвходовото „логическо НЕ-И“ сменя своето състояние на ниско логическо ниво. Следователно стойността на втория брояч **U14** се увеличава с единица.

3.2.5.3 Очертание „Програмираме осцилатор“

Схемният символ на програмируемия осцилатор DS1085 е разположен на 2-ри лист (фиг. 3.32) от принципната електрическа схема в очертанието „PROGRAMMABLE OSCILLATOR“ и е означен с **U8**. Осцилаторът се захранва от **+5VD** спрямо **GNDD**, като към захранващия му пин (**VCC**) е включен отделящият кондензатор **C17**.

Интегралната схема се програмира от микроконтролера посредством I²C интерфейс. **U8** е закачен към комуникационните линии **SDA** и **SCL** съответно с пин 7-ми и пин 8-ми. Освен това другите контролни изводи **CTRL0** и **CTRL1** са свързани съответно към сигналните линии **PWRDWN** и **OSCOE**, които се управляват от ATmega2561.

Главният изход (**OUT1**) на програмируемия осцилатор се използва като тактов контролен сигнал за адресния брояч, защото може да се регулира прецизно в широкия обхват от 8.1kHz до 133MHz⁽¹³⁾, което позволява прецизно регулиране на честотата на генерирания сигнал. Референтният изход (**OUT0**) се използва като източник на тактов сигнал, който задава непрецизно скоростта на актуализиране на цифрово-аналоговия преобразувател на DDS системата.

3.2.5.4 Очертание „Преходник последователна-паралелна комуникация“

Преходникът MCP23017 е изобразен на фиг. 3.33 и означен с **U10**. Интегралната схема се захранва от **+5VD** спрямо **GNDD**, като към захранващия му пин (**VDD**) е включен отделящият кондензатор **C19**.

Компонентът се настройва и управлява от микроконтролера посредством I²C интерфейс. Преходникът е закачен към комуникационните линии **SDA** и **SCL** съответно с пин 13-ти и пин 12-ти. Понеже I²C адресните му пинове (**A0** ÷ **A2**) са заземени (към **GNDD**), устройството придобива адреса „0100 000“⁽¹⁴⁾. Изводът **RESET** се използва за рестартиране на преходника при подаване на ниско логическо ниво и е свързан към пин **PF1** на ATmega2561. Изводите **INTA** (20-ти пин) и **INTB** (19-ти пин) на **U10** не се използват.

Пиновете **GPB0** ÷ **GPB7** на **U10** са свързани към младшите входно-изходни пинове на SRAM паметта **U9** - съответно към **I/O₀** ÷ **I/O₇**, докато пиновете **GPA0** ÷ **GPA7** - съответно към **I/O₈** ÷ **I/O₁₅**. Следователно посредством порт В на преходника се записват данни в ниския байт (младшите 8 бита) на паметта **U9**, а чрез порт А - във високия байт (старшите 8 бита).

3.2.5.5 Очертание „ЦАП“

Това очертание също е разположено на 2-ра страница (фиг. 3.33) от принципната електрическа схема. То включва цифрово-аналоговия преобразувател на DDS системата, означен с **U11**, както и необходимите му спомагателни елементи за правилното му функциониране.

ЦАП-ът е захранен от две отделни захранвания. Цифровата му част е захранена от **+5VD** спрямо **GNDD**, а аналоговата - **+5VA** спрямо **GNDA**. Към захранващите пинове **DVDD** и **AVDD** на интегралната схема са свързани по два отделящи кондензатори - съответно **C21** и **C22** към цифровото и **C23** и **C24** към аналоговото. **C21** и **C23** са танталови кондензатори с номинален капацитет 10uF, които изглеждат допълнително захранващите напрежения, шунтирайки нискочестотните смущения към общия проводник. Наложително е двете земи да се обединят в една обща точка, наречена звездна точка (от англ.: „star ground“)⁽⁵⁾, спрямо която се измерват всички напрежения в електрическата схема. Връзката между цифровата земя (**GNDD**) и аналоговата земя (**GNDA**) в звездна точка е изобразена в очертанието „DDS SYSTEM DAC“ на фиг. 3.33 под преобразувателя **U11**, защото физически тази връзка трябва да бъде направена близо до ЦАП-а на печатната платка⁽⁵⁾.

Цифровият интерфейс на преобразувателя се състои от цифровите входове (**DB0** ÷ **DBn**), които определят резултацията му. Тези входове са различен брой според избрания модел ЦАП от серията TxDAC на Analog Devices, като на схемата е показан представителят с най-много входни пинове (14 на брой) - AD9764. Представителите на TxDAC фамилията са пиново съвместими, защото най-старшите им битове винаги са разположени на 1-вия пин, а по-младшите битове са последователно разположени след най-старния. Цифровият интерфейс на **U11** включва и два допълнителни извода:

- **CLOCK** (28-ми пин): към този извод трябва да се свърже източник на периодичен тактов сигнал. При всеки нарастващ фронт на тактовия сигнал преобразувателят запазва моментната стойност, подадена на входните му пинове (**DB0** ÷ **DBn**), във вградени в интегралната схема тригери, след което запазената стойност се преобразува в аналогова изходна величина. Към този извод е свързан референтният изход (**OUT0**) на програмируемия осцилатор **U8**.
- **SLEEP** (15-ти пин): при подаване на високо логическо ниво на този извод, ЦАП-ът „заспива“, при което съществено се намалява консумацията му. Този извод е свързан към сигналната линия **PWRDWN**, контролирана от микроконтролера.

Аналоговият изход на ЦАП-а се състои от два комплементарни токови изходи (**IOUTA** и **IOUTB**). Тяхната абсолютна сума винаги е равна на пълният изходен ток (I_{OUTFS}), който от своя страна зависи от зададения референтен ток (I_{REF}). Взаимозависимостите между споменатите величини са описани в каталожните данни⁽⁹⁾ чрез следните изрази:

$$I_{OUTA} = (DAC\ CODE / 16384) \times I_{OUTFS} \quad [3.19]$$

$$I_{OUTB} = (16383 - DAC\ CODE) / 16384 \times I_{OUTFS} \quad [3.20]$$

$$I_{OUTFS} = 32 \times I_{REF} \quad [3.21]$$

Референтният ток (I_{REF}) от своя страна се задава чрез външни хардуерни елементи и се описва с формулата [3.22]. Елементът **U12** външно задава опорното напрежение (V_{REFIO}) +1.25V на ЦАП-а. Изводът **REFLO** (16-ти пин) на ЦАП-а трябва да бъде свързан към аналоговото захранване (+5VA), за да се изключи вътрешният източник на опорно напрежение на **U11**. Освен това към **FS_ADJ** е включен резисторът **R5**, който изпълнява ролята на R_{SET} в [3.22]:

$$I_{REF} = V_{REFIO} / R_{SET} = 1.25V / 2k\Omega = 0.625mA \quad [3.22]$$

Замествайки в [3.21], става ясно, че сумарният ток на двета изхода (I_{OUTFS}) е равен на 20 mA.

Към изводите **COMP1** и **COMP2** на ЦАП-а са свързани керамичните кондензатори **C26** и **C25**, имащи номинален капацитет 100nF, по препоръка на производителя в каталожните данни⁽⁹⁾.

3.2.6 Блок „Изходна аналогова схемотехника“

В този блок се включва изходната аналогова схемотехника, разположена изцяло на 4-тата страница (фиг. 3.35). Схемата представлява четиристъпален усилвател, съставен от операционните усилватели AD8021.

Първото стъпало е съставено от операционния усилвател, означен с **U16** на фиг. 3.35. **U16**, както и всички останали операционни усилватели AD8021 на схемата, са захранени от двойното захранващо напрежение $\pm 12\text{VA}$ спрямо **GNDA**, като и двете захранвания са изгладени с отделящи кондензатори (за **U16** това са **C34** и **C31**). Между 5-ти пин на **U16** и отрицателното захранване -12VA е включен кондензаторът **C32**. Последният е компенсационен кондензатор, сложен по препоръка от производителя. Изборът на номиналния капацитет на компенсационния кондензатор е извършен с помощта на таблица 6, дадена на фиг. A.7, от каталожните данни⁽¹⁵⁾ на AD8021. Както се забелязва, при по-големи стойности на компенсационния капацитет и по-малък коефициент на усилване честотната лента и шумовите свойства на усилвателя се подобряват, но за сметка на това се намалява скоростта на промяна на сигнала (от англ.: „slew rate“). За увеличаване на скоростта е необходимо да се намали капацитета на компенсационния кондензатор или изцяло да се избегне слагането му, при което се влошават шумовите свойства на операционния усилвател. Производителят отбелязва, че компенсационният кондензатор трябва да е SMD кондензатор с висококачествен COG или NPO диелектрик.

Първото стъпало представлява диференциален усилвател, конвертиращ двета токови изхода на ЦАП-а на DDS системата в единствен сигнал по напрежение. Стъпалото е проектирано по схемата образец за диференциален-към-единствен изход, дадена в каталожните данни на AD9764 (стр. 14 от (9)). Двойките резистори **R7-R8** и **R9-R10** образуват сумарни съпротивления от по 25Ω . При извършване на преобразуване от ЦАП-а и следователно протичане на комплементарните токове върху резисторите двойки се отделят падове на напрежение, равни на:

$$V_{OUTA} = I_{OUTA} \times 25\Omega \quad [3.23]$$

$$V_{OUTB} = I_{OUTB} \times 25\Omega \quad [3.24]$$

Понеже максималният сумарен ток на ЦАП-а е 20mA, максималният пад на напрежение, отделено върху една от двойките резистори е с големина 0.5V.

Получените напрежения се подават на двета входа на усилвателя през резисторите **R11** и **R12**. Тъй като резисторът **R17** е свързан с цел образуване на отрицателна обратна връзка, усилването на първото стъпало е:

$$GAIN_1 = \frac{R17}{R11} = \frac{1k\Omega}{510\Omega} \approx 1.96 \quad [3.25]$$

Следователно изходното напрежение на първото стъпало е с максимална амплитуда:

$$V_{OUT} = \pm 0.5V \times GAIN_1 = \pm 0.5V \times 1.96 = \pm 0.98V \quad [3.26]$$

Второто усилвателно стъпало се използва за изменя амплитудата на сигнала. Стъпалото може да бъде хардуерно конфигурирано по два различни начини според желания начин за изменя на амплитудата - електронно или ръчно. Двата начина на конфигурация се разглеждат по-надолу. Самото стъпало е образувано от операционния усилвател AD8021, включен в стандартна схема на инвертиращ усилвател и означен с **U22** на фиг. 3.35. Интегралната схема се захранва от двойното аналогово захранване **±12VA**, като и към двета му захранващи извода са включени отделящите кондензатори **C44** и **C46**. Кондензаторът **C45** е компенсационен, като функциите и изборът му бяха коментирани. Двойката резистори **R33** и **R34** имат сумарно съпротивление 2.5kΩ и участват в отрицателна обратна връзка за **U22**. Към инвертиращия вход (2-ри извод) се подава генерирания сигнал за усилване. Към неинвертиращия вход е свързан резисторът **R32**, който се използва за симетриране на входовете на AD8021 по постоянен ток⁽²⁸⁾, както и източникът на опорно напрежение AZ431AN⁽²⁹⁾, означен с **U21**. Източникът определя постояннотоковия потенциал от +2.5V на неинвертиращия вход на **U22**, като добавеният потенциал е необходим за отместяване на изхода на усилвателя по постоянен ток с +2.5V.

Както беше споменато, второто усилвателно стъпало може да се настрои или за ръчна изменя на амплитудата на генерирания сигнал, или за електронна изменя (т.е. контролирана от микроконтролера) е необходимо да се включат **R19**, **R20**, **R23**, **R31**, поне един от JFET транзисторите **Q1** ÷ **Q3**, както и всички елементи, включени в очертанието „SIGNAL GAIN CONTROL“. **RV1** и **JP1** не участват в електронния метод за изменя на

амплитудата на генерирания сигнал. При включване на **R31** усилването на второто стъпало е равно на:

$$GAIN_2 = \frac{R33 + R34}{R31} = \frac{2.5 k\Omega}{1 k\Omega} = 2.5 \quad [3.27]$$

Амплитудата на генерирания сигнал се изменя посредством JFET транзисторите MMBF5485, които се използват като променливи резистори⁽³⁰⁾. Идеята е транзисторът да бъде използван в омичния си регион, където поведението му е на линейно съпротивление. Използването на JFET транзистор в омичния му регион се постига чрез подаване на малко напрежение дрейн-сорс (което в случая е с максимална амплитуда $\pm 0.98V$), за да не се насити транзисторът, и чрез контролиране на напрежението гейт-сорс (V_{GS}). Поради силициевата структура на JFET транзисторите минималното постижимо съпротивление дрейн-сорс ($r_{DS(ON)}$) се постига при нулево напрежение гейт-сорс (V_{GS}). С намаляване на напрежението гейт-сорс (V_{GS}) транзисторът се насища с по-малко напрежение дрейн-сорс и съпротивлението му дрейн-сорс (r_{DS}) се увеличава. Следователно JFET транзисторът се държи като променливо съпротивление, контролирано с напрежение. В схемата от фиг. 3.35 се използват JFET транзисторите MMBF5485⁽³¹⁾, които имат удобни за приложението параметри. Използвайки изходната характеристика на транзистора от каталожните данни⁽³¹⁾ може приблизително да се изчисли съпротивлението дрейн-сорс ($r_{DS(ON)}$) посредством закона на Ом за малка област от характеристиката:

$$r_{DS(ON)} = \frac{\Delta V_{DS}}{\Delta I_{DS}} \approx 150 \Omega \quad [3.28]$$

Направеното изчисление в израза [3.28] е само ориентировъчно, понеже характеристиките на реалните транзистори варират изключително много.

Съпротивлението **R23** и JFET транзистора **Q1** са включени в схема тип делител на напрежение по отношение на изходния сигнал на първото усилвателно стъпало. Амплитудата на генерирания сигнал не се контролира чрез изменя амплитудата на второто стъпало, а чрез изменя на подадения от първото стъпало сигнал. При насищане на транзистора съпротивлението му е максимално (в областта на десетки и стотици килооми) и генеририаният сигнал е почти изцяло предаден на входа на второто стъпало. От друга страна, при минимално съпротивление дрейн-сорс ($r_{DS(ON)}$) предадената част от сигнала е приблизително:

$$V_{IN2} = V_{OUT1} \times \frac{r_{DS(ON)}}{R23} = \pm 0.98V \times \frac{150 \Omega}{2k\Omega} \approx \pm 0.07V \quad [3.29]$$

Макар че в схемата са разположени няколко транзистора (**Q1** ÷ **Q3**), само един MMBF5485 е необходим за нормалното ѝ функциониране. При включване на повече от един транзистор паралелно минималното съпротивление дрейн-сурс ($r_{DS(ON)}$) намалява. За сметка на това се намалява максималното съпротивление и се увеличава сумарния паразитен капацитет на транзисторите, което може да повлияе на предавания сигнал, защото паразитният капацитет и съпротивлението **R23** образуват нискочестотен филтър. Все пак броя на включените транзистори трябва да се установи експериментално.

JFET транзисторите са включени в схема с обратна връзка, проектирана по образца, представен на фиг. 6 от документа (30). Внасянето на малка обратна връзка значително намалява изкривяванията. Обратната връзка се осъществява от резисторите **R19** и **R20**.

Контролирането на напрежението гейт-сурс (V_{GS}), приложено на JFET транзисторите се извършва от схемата, разположена в очертанието „SIGNAL GAIN CONTROL“ на фиг. 3.35. Интегралната схема MCP4716⁽³²⁾, означена с **U18**, представлява цифрово-аналогов преобразувател, управляван през I²C интерфейс. Преобразувателят се захранва от аналоговото напрежение +5V_A, като на захранващия му извод (3-ти) е включен отделящият кондензатор **C37**. Като опорно напрежение MCP4716 може да използва захранващото напрежение, подадено на извода **VDD**, поради което изводът **VREF** не се използва и е оставен несвързан. Използвайки захранващото напрежение +5V_A като опорно, изходът на 10-битовия ЦАП може да заеме стойности в интервала 0V ÷ 5V. Това напрежение обаче не е изгодно за управление на гейтовете на JFET транзисторите. Затова изходът на ЦАП-а е подаден като вход на операционния усилвател MIC6211⁽³³⁾, означен на схемата с **U20**. Интегралната схема е захранена с двойното аналогово напрежение ±12V_A, като към захранващите изводи са свързани отделящите кондензатори **C40** и **C41**. Този операционен усилвател е свързан в схема тип „изваждащ усилвател“, при който изходният сигнал е разликата на двета входни сигнала:

$$V_{OUT} = V_{+IN} - V_{-IN} \quad [3.30]$$

Тъй като и четирите съпротивления, участващи в схемата - **R25**, **R26**, **R27**, **R28**, са с еднакво съпротивление, усилването на схемата е единица. Следователно единствената функция, която изпълнява **U20**, е отместването на контролния сигнал за JFET транзисторите от **U18** с -5V. Тогава контролният сигнал, подаден на гейтовете на транзисторите, може да заеме стойности в интервала -5V ÷ 0V, който е удобен интервал за управление на JFET транзисторите.

Изходният сигнал на второто стъпало, означен с табелата **OCAL**, е подаден обратно на микроконтролера, свързвайки го към пин **PF3**. Към този пин вътрешно в микроконтролера е свързан един от входните канали (**ADC3**) на вграденото АЦП. Използвайки вграденото си АЦП, микроконтролерът може да настрои прецизно изходното напрежение на второто стъпало чрез конфигуриране на подаденото от **U18** контролно напрежение гейт-сорс на транзисторите до постигане на желаната амплитуда на генерирация сигна.

В случай, че потребителят иска да изменя амплитудата на генерирания сигнал ръчно, е необходимо да се свържат само елементите **RV1** и **JP1**. Останалите елементи, които се използват за електронно контролиране на амплитудата - **R19, R20, R23, R31, Q1 ÷ Q3**, както и компонентите в очертанието „**SIGNAL GAIN CONTROL**“, не трябва да се включват в схемата. Джъмперът **JP1** е представен символично на схемата, като на реалната платка той се „включва“ чрез окъсяване медните плохи на **R23**. **RV1** представлява механичен реостат, който се използва за изменя на усилването на второто стъпало, като по този начин се променя и амплитудата на генерирания сигнал.

Резултатният усилен от второто стъпало сигнал се подава на следващото трето стъпало. Последното е съставено от операционния усилвател AD8021, означен с **U17**. Усилвателят също е захранен от двойното захранващо напрежение **±12VA** и има компенсационен кондензатор, означен с **C35**. Резисторът **R16** е свързан към неинвертиращия вход на усилвателя **U17** и се използва за симетриране на входовете на усилвателя по постоянен ток⁽²⁸⁾. Усилвателят е свързан в схема тип „сумиращ усилвател“ с коефициент на усилване единица. Функцията на това стъпало е да добави желаното постояннотоково отместване към генерирания сигнал. Постояннотоковото отместване се генерира от цифрово-управляемия потенциометър MCP4551⁽³⁴⁾, означен на схемата с **U15**. Интегралната схема се захранва от аналоговото напрежение **+5VA** спрямо **GNDA**, като към захранващия ѝ извод е включен отделящият кондензатор **C30**. Понеже потенциометърът не може да работи с напрежения извън обхвата на захранващото (т.е. извън интервала $0V \div 5V$), между двета крайни извода е включено напрежение с големина $5V$. Изкараното от средния извод **P0W** напрежение, което приема стойности в интервала $0V \div 5V$, се подава като един от сумираните от третото стъпало сигнали. Другият сумиран входен сигнал е генерираният от DDS системата сигнал. Резултатният изход на третото стъпало представлява желаният генериран сигнал, отместен с желаното постояннотоково ниво.

Последното усилвателно стъпало е съставено от операционния усилвател **U19**, захранен с двойното захранване $\pm 12\text{VA}$. Понеже и този усилвател е AD8021, към него е включен компенсационният кондензатор **C39**. Четвъртото стъпало представлява схема тип „изваждащ усилвател“, при който изходният сигнал е разликата от входните, както е показано в израз [3.30]. Следователно изходният сигнал на стъпалото представлява генерираният от DDS системата сигнал, от който е извадено постояннотоковото ниво $+5\text{V}$. Има две причини, поради които в това стъпало се изваждат статично 5V . Първата е наличието на допълнително постояннотоково отместване с големина $+2.5\text{V}$, добавено към сигнала във второто стъпало от **U21**. От друга страна, цифровият потенциометър **U15** от третото стъпало може да добавя само положително постояннотоково отместване, понеже не може да бъде захранен от двойно напрежение. Понеже средната точка на потенциометъра подава $+2.5\text{V}$ на входа на третото стъпало, е необходимо да се извадят допълнителни $+2.5\text{V}$ от изходния сигнал на третото стъпало. Това означава, че потенциометърът **U15** ефективно отмества генерирания от DDS системата сигнал с постояннонотоково напрежение в интервала $\pm 2.5\text{V}$. Също така четвъртото стъпало има коефициент на усилване равен на 2.

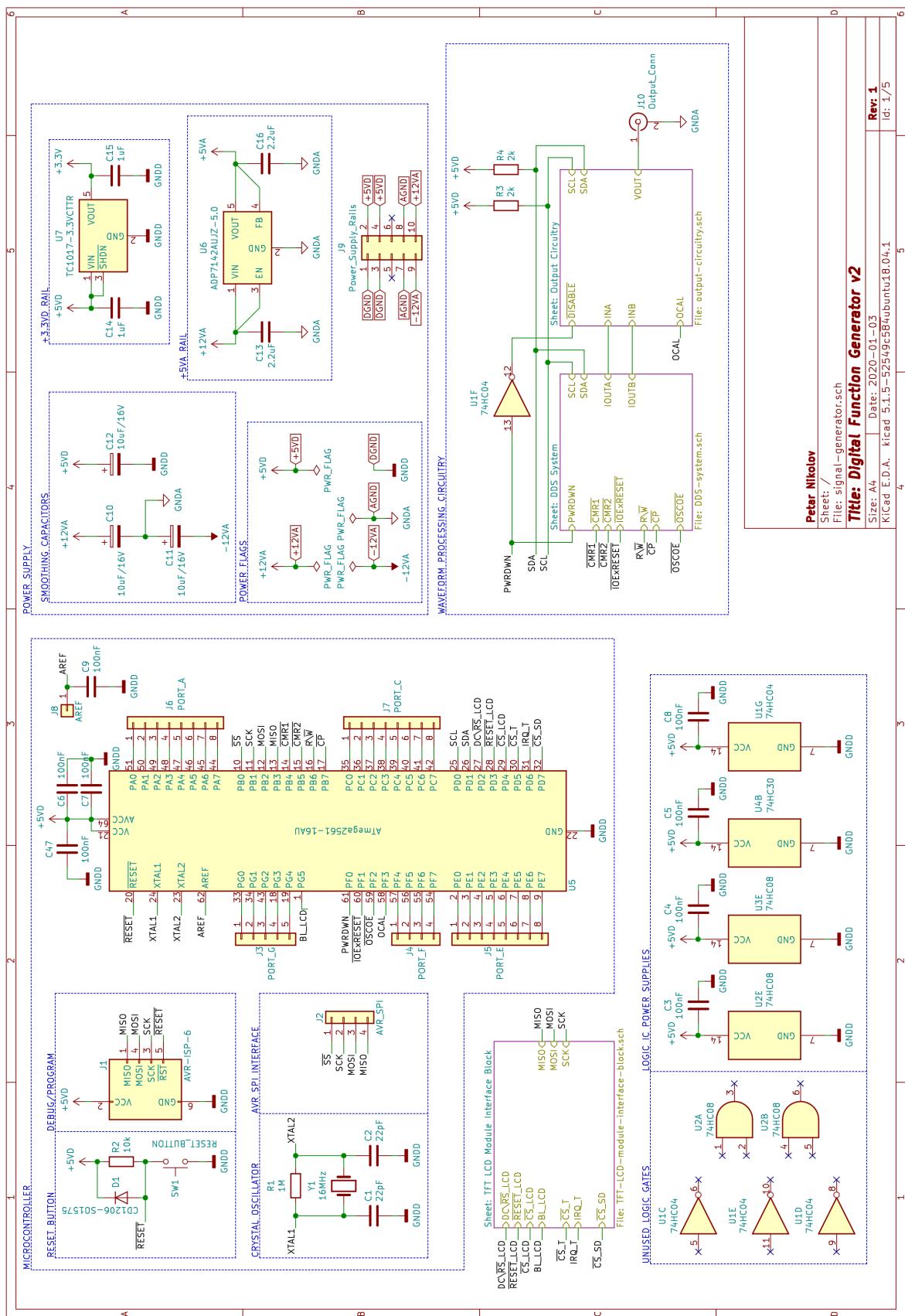
Резултатният обработен от усилвателя изходен сигнал е подаден на BNC конектора **J10** от фиг. 3.32. Чрез **J10** потребителят може да използва генерирания сигнал за свои цели.

3.3 I²C адресация

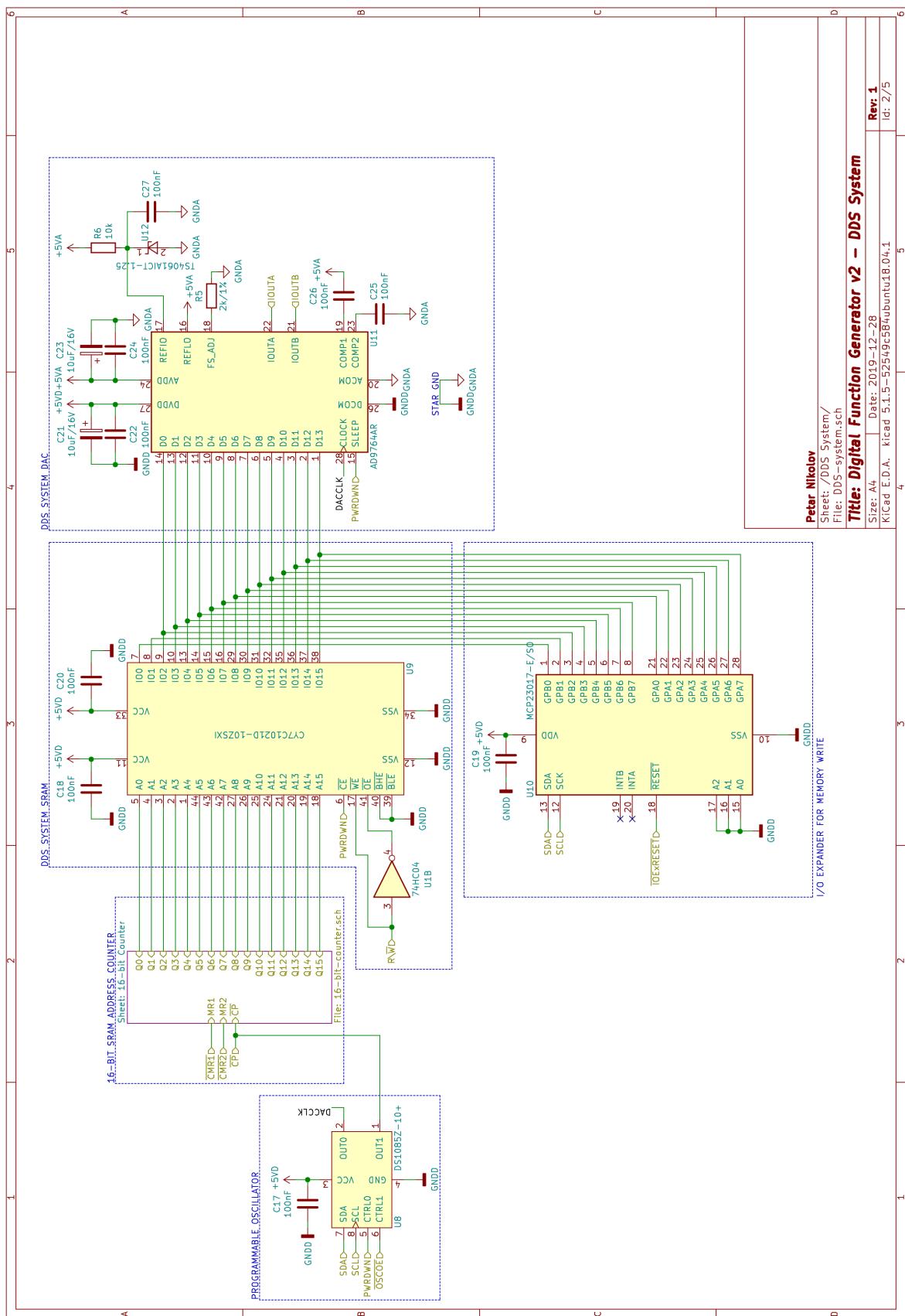
В таблица 3.31 са дадени I²C адресите на всички I²C устройства, участващи в принципната електрическа схема на генератора на сигнали.

УСТРОЙСТВО	ОЗНАЧЕНИЕ	I ² C АДРЕС
DS1085Z-10+	U8	1011 000
MCP23017	U10	0100 000
MCP4551-103E	U15	0101 110
MCP4716-A0T	U18	1100 000

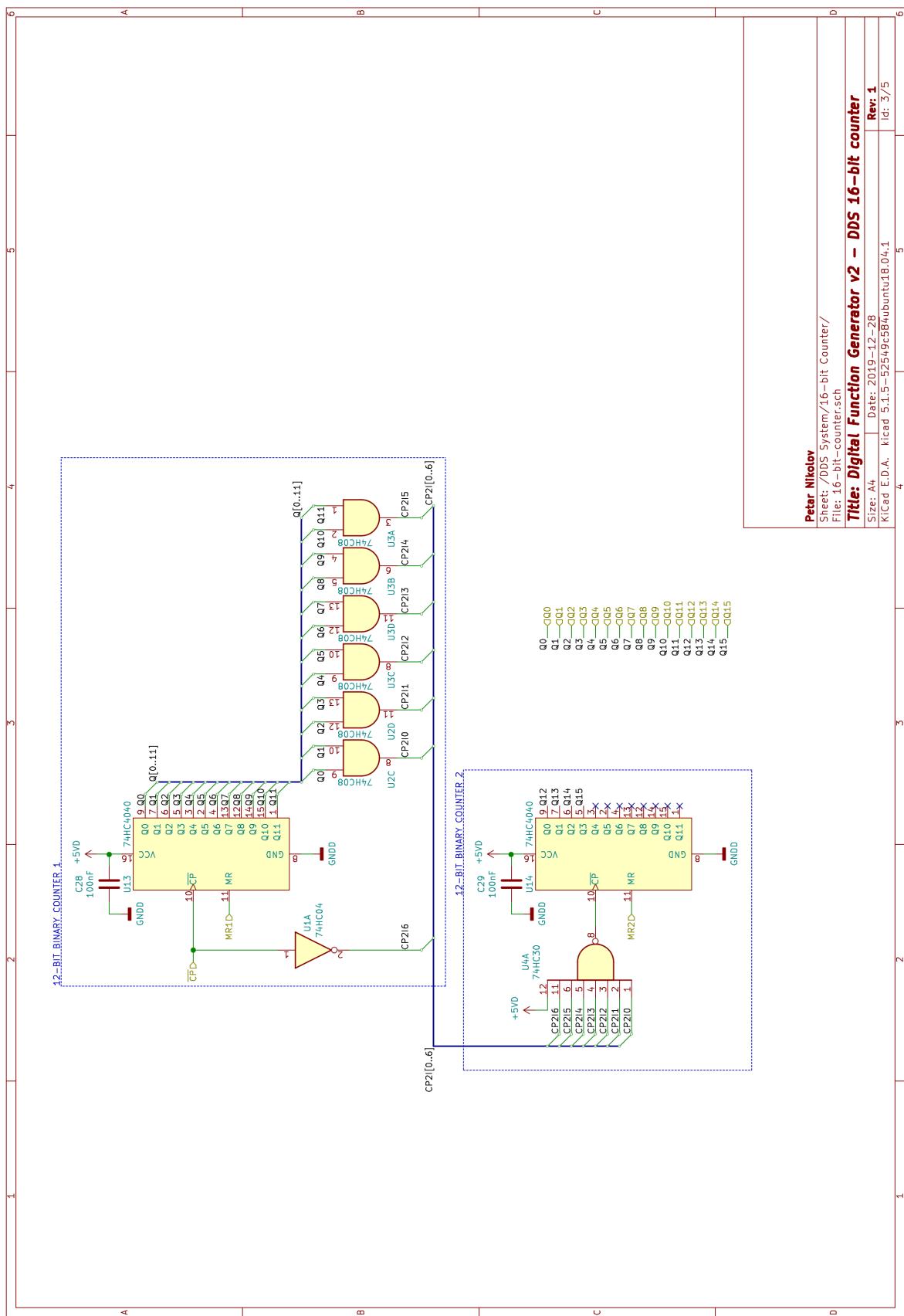
таблица 3.31: I²C адреси на I²C устройствата



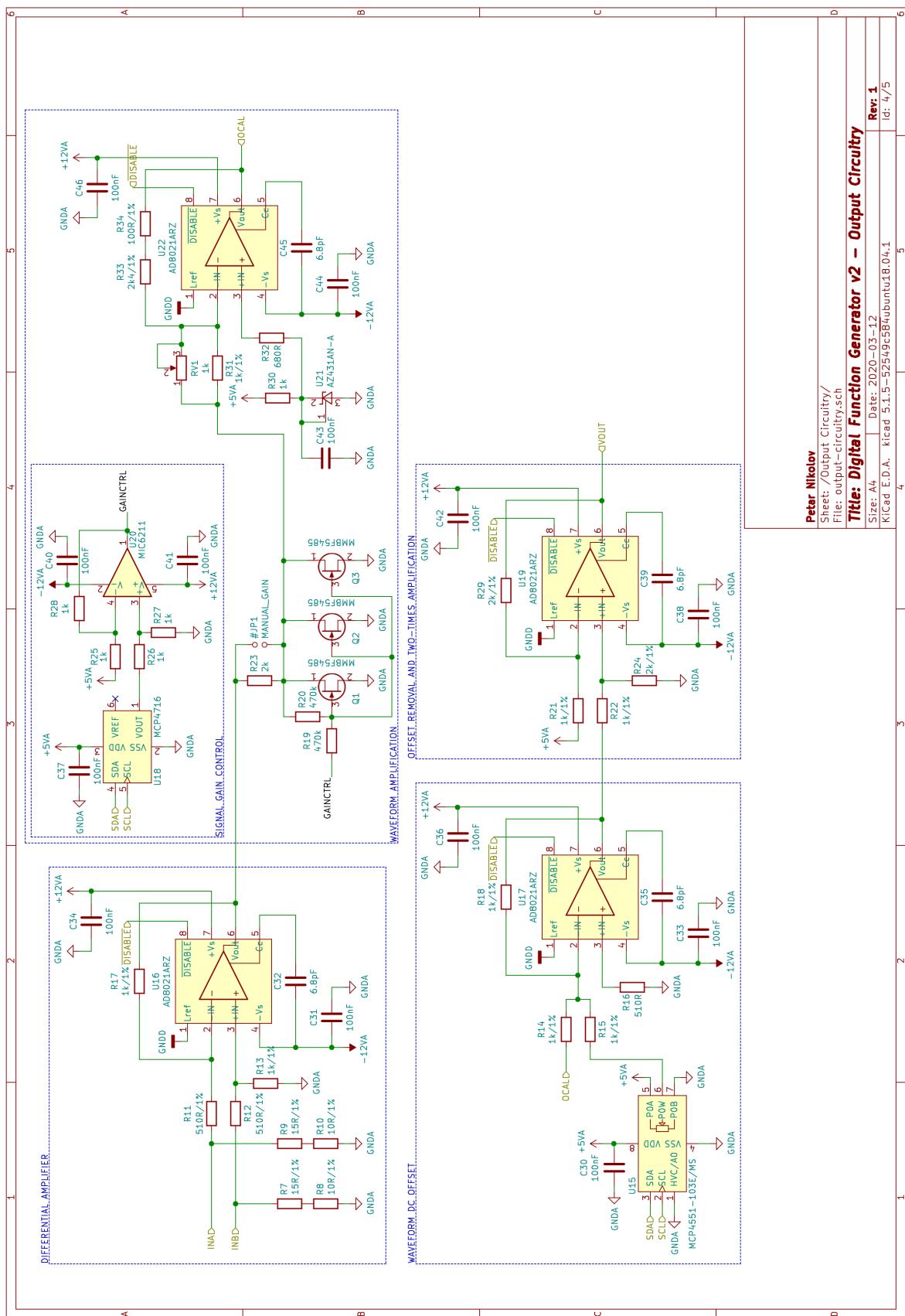
фиг. 3.32: Принципна електрическа схема на цифров генератор на сигнали - 1-ва страница



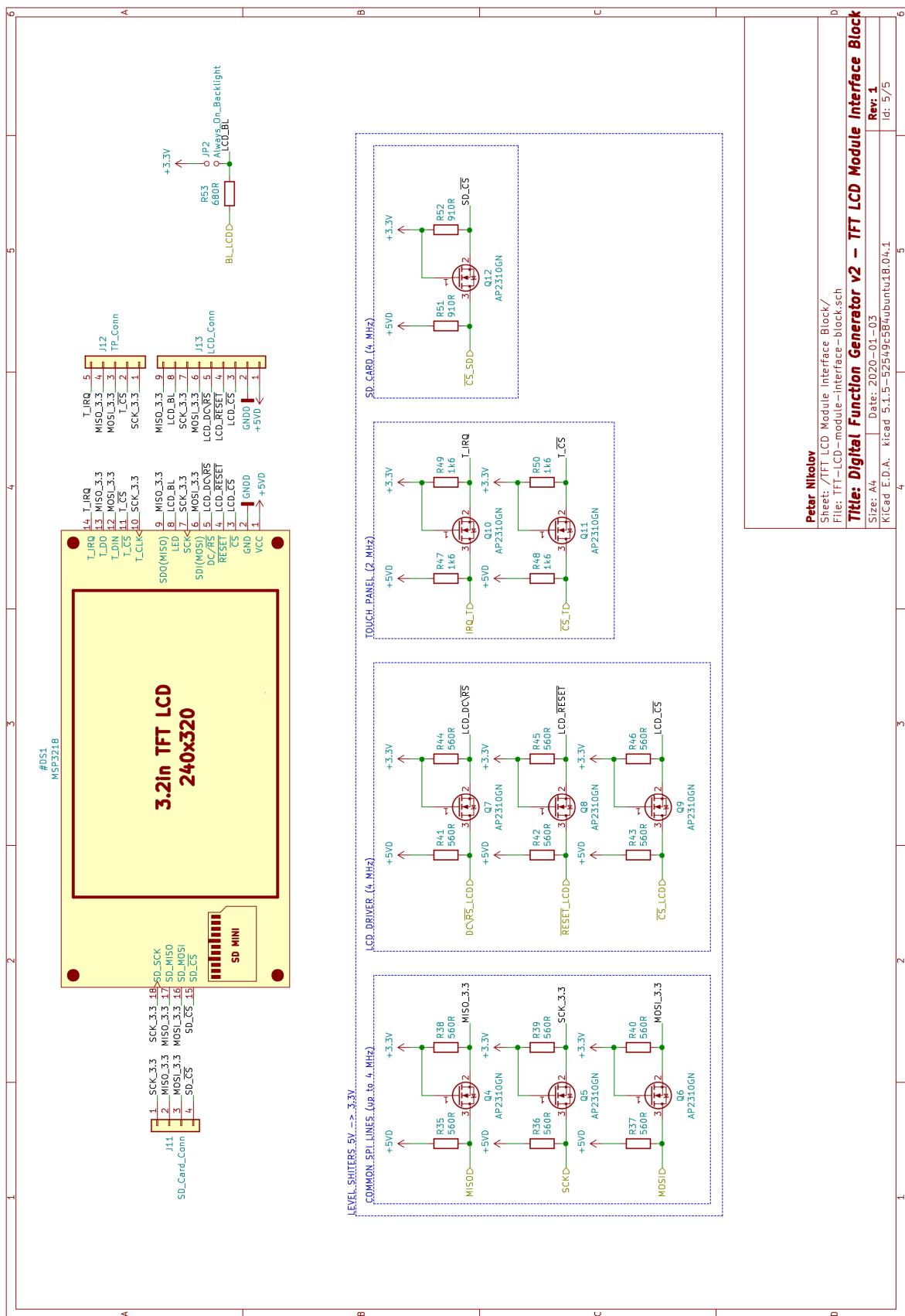
фиг. 3.33: Принципна електрическа схема на цифров генератор на сигнали - 2-ра страница



фиг. 3.34: Принципна електрическа схема на цифров генератор на сигнали - 3-та страница



фиг. 3.35: Принципна електрическа схема на цифров генератор на сигнали - 4-та страница



фиг. 3.36: Принципна електрическа схема на цифров генератор на сигнали - 5-та страница

ГЛАВА IV

Проектиране на графични оригинали на печатни платки

4.1 Корпуси на използвани елементи

В таблица 4.1 са дадени корпусите на всички използвани в устройството елементи. Таблицата също съдържа и означенията на компонентите на принципната електрическа схема, както и начина на монтаж (SMD или THT).

4.2 Параметри на проводящите писти

При проектиране на печатната платка са използвани различни медни проводящи писти с различни параметри. Нуждата от проводящи писти с различни характеристики се обуславя от необходимостта за пренасяне на различни видове сигнали - аналогови или цифрови, бързоскоростни или нискоскоростни и т.н. Затова в процеса на проектиране на платката в KiCAD са използвани различни „класове“ проводящи писти, като конкретните писти, разположени на платката, се причисляват към един от „класовете“, с добивайки се с неговите параметри. Използваните „класове“, както и параметрите им, са представени нагледно в таблица 4.2, като размерите са дадени и в милиметри (mm), и в мерната единица мил (mil). Параметрите на проводящите писти, изброени в таблицата са:

- **Clearance:** минималното разстояние между проводящата писта и други проводящи писти/медни подложки (на компонентите);
- **Track width:** широчината на проводящата писта;
- **Via size:** диаметърът на проводящи проходи (via);
- **Via drill:** диаметърът на вътрешните дупки на проводящите проходи (via).

За пренасяне на високоскоростни сигнали се използват проводящи писти с по-големи размери за постигане на по-висока проводимост и по-добра изолация на проводника, пренасящ сигнала. За пренасяне на аналогови сигнали се използват проводящи писти с по-големи размери спрямо пистите, пренасящи цифрови сигнали, защото аналоговата схемотехника е по-уязвима към външни смущения спрямо цифровите схеми. Друга важна особеност е, че диаметърът на дупките на проводящите проходи (via) е винаги два пъти по-малък от диаметъра на самите проводящи проходи (via).

Елемент	Означение/я	Монтаж	Корпус
Керамични Кондензатори	C1 ÷ C9, C13 ÷ C20, C22, C24 ÷ C47	SMD	C0805
Алуминиеви Електролитни Кондензатори	C10 ÷ C12	SMD	Конд. Електр. 4mm x 5.4mm
Танталови Електролитни Кондензатори	C21, C23	SMD	EIA3216-18
CD1206-S01575	D1	SMD	D1206
Резистори	R2 ÷ R53	SMD	R0805
Потенциометър	RV1	THT	RK163 (единичен)
MMBF5485	Q1 ÷ Q3	SMD	SOT-23
AP2310GN	Q4 ÷ Q12	SMD	SOT-23
74HC04	U1	SMD	SO14
74HC08	U2, U3	SMD	SO14
74HC30	U4	SMD	SO14
74HC4040	U13, U14	SMD	SO16
ATmega2561-16AU	U5	SMD	TQFP-64
ADP7142AUJZ-5.0	U6	SMD	TSOT-23-5
TC1017-3.3VCTTR	U7	SMD	SOT-23-5
DS1085Z-10+	U8	SMD	SO8
CY7C1021D-10ZSXI	U9	SMD	TSOP-44-II
MCP23017-E/SO	U10	SMD	SO28W
AD9764AR	U11	SMD	SO28W
TS4061AICT-1.25	U12	SMD	SOT-323 (SC-70)
MCP4551-103E/MS	U15	SMD	MSOP-8
AD8021ARZ	U16, U17, U19, U22	SMD	SO8
MCP4716	U18	SMD	SOT-23-6
MIC6211	U20	SMD	SOT-23-5
AZ431AN-A	U21	SMD	SOT-23
Кварцов кристал	Y1	THT	HC-49S
Тактилен бутон	SW1	THT	DIP switch 6mm x 4.3mm
Пинови съединители	J1 ÷ J9, J11 ÷ J13, JP2	THT	Pin Header Straight (Pitch 2.54mm)
Изходен конектор	J10	THT	BNC

таблица 4.1: Корпуси на компонентите на проектирания генератор

Name	Clearance, mm(mil)	Track width, mm(mil)	Via size, mm(mil)	Via drill, mm(mil)
Analog HS Signal	0.1905(7.5)	0.381(15)	0.6096(24)	0.3048(12)
Analog Signal	0.127(5)	0.254(10)	0.6096(24)	0.3048(12)
Analog Supply	0.127(5)	0.508(20)	0.8128(32)	0.4064(16)
Bypass Cap	0.127(5)	0.254(10)	0.6096(24)	0.3048(12)
Compensation Cap	0.127(5)	0.508(20)	0.8128(32)	0.4064(16)
Digital HS Signal	0.127(5)	0.1905(7.5)	0.6096(24)	0.3048(12)
Digital Signal	0.127(5)	0.127(5)	0.508(20)	0.254(10)
Digital Supply	0.127(5)	0.254(10)	0.6096(24)	0.3048(12)
Digital UHS Signal	0.1905(7.5)	0.254(10)	0.8128(32)	0.4064(16)
I2C	0.127(5)	0.1905(7.5)	0.6096(24)	0.3048(12)
SPI	0.127(5)	0.1905(7.5)	0.6096(24)	0.3048(12)
VOUT	0.1905(7.5)	0.508(20)	0.8128(32)	0.4064(16)

таблица 4.2: Класове проводящи писти

Следва описание на използваните класове проводящи писти:

- **Digital UHS Signal:** използва се за писти, които потенциално може да пренасат изключително високоскоростни цифрови сигнали (в порядъка на десетки MHz).
- **Digital HS Signal:** използва се за писти, които пренасят високоскоростни цифрови сигнали (в порядъка на няколко MHz).
- **Digital Signal:** пистите от този клас пренасят всякакви невисокоскоростни цифрови сигнали.
- **I2C:** този клас се прилага единствено на проводниците, използвани за изграждане на I²C комуникация (SCL и SDA).
- **SPI:** този клас се прилага единствено на проводниците, използвани за изграждане на SPI комуникация (MISO, MOSI и SCK).
- **Analog HS Signal:** прилага се на писти, пренасящи високоскоростни (в порядъка на няколко MHz) аналогови сигнали.
- **Analog Signal:** използва се за писти, пренасящи невисокоскоростни аналогови сигнали.
- **VOUT:** включва единствено пистата, свързваща изхода на четиристъпалния аналогов усилвател с изходния BNC конектор.
- **Digital Supply:** включва проводящи писти, които захранват цифрови интегрални схеми и компоненти.

- **Analog Supply:** включва проводящи писти, които захранват аналогови интегрални схеми и компоненти.
- **Bypass Cap:** изполва се за писти, които свързват отделящи керамични кондензатори с други компоненти.
- **Compensation Cap:** проводящи писти, които свързват поставените за операционните усилватели AD8021 компенсационни кондензатори (виж т.3.2.6).

Важно е да се отбележи, че на печатната платка са разположени проводящи писти, които не са включени в нито един от изброените „класове“ писти и имат размери, които не са описани в таблица 4.2. Причините за наличието на писти с уникални размери са коментирани по-надолу в тази глава.

4.3 Особености на слоевете на печатната платка

Графичните оригинали на проектираната печатна платка са представени на фиг. 4.3, фиг. 4.4, фиг. 4.5 и фиг. 4.6. Платката е съставена от четири отделни медни слоя:

- **1-ви слой (Горен):** на този слой са разположени сигналните проводящи писти, осъществяващи нормалното функциониране на устройството, свързвайки използваните интегрални схеми и други компоненти помежду си. Горният слой е показан на фиг. 4.3;
- **2-ри слой (Захранващ):** захранващият слой осигурява необходимите, максимално предпазени от външни смущения, захранващи напрежения на всяка интегрална схема. Захранващият слой е представен на фиг. 4.4;
- **3-ти слой (Земя):** на този слой са разположени общите проводници (заземяващите площи) на двете отделни захранвания (цифровото и аналоговото). Този слой е показан на фиг. 4.5;
- **4-ти слой (Долен):** и този слой, както горния (1-ви), се използва за разположение на проводящи писти, пренасящи сигнали. Долният слой е показан на фиг. 4.6.

SMD компонентите са разположени и от двете страни на платката. От друга страна, елементите с ТНТ монтаж са разположени на горната страна на печатната платка, като на долната страна се намират съответните им спойки.

4.3.1 Сигнални слоеве (1-ви и 4-ти)

На тези два слоя са разположени всички сигнални проводящи писти, спойките на ТНТ елементите и медните подложки на SMD елементите. Горният и долният слой осъществяват свързаността между всички електронни компоненти, съставящи генератора на сигнали, и съответно гарантират нормалното и правилното функциониране на използваните интегрални схеми. В тази точка са обсъдени само най-важните взети решения и най-характерните особености на двета сигнални проводящи слоя.

Подходът, предприет при разположението на елементите върху печатната платка, възможно най-много скъсява дължината на проводящите писти и възможно най-малко излага интегралните схеми на външни смущения от околната среда. При проектирането на платката първоначално са разположени елементите на функционалните блокове, съставящи DDS системата на генератора. ЦАП-ът, SRAM паметта, адресният брояч и др. са сложени възможно най-близко помежду си по средата на платката. Пистите свързващи изброените елементи пренасят най-високоскоростните цифрови сигнали.

От друга страна, микроконтролерът, периферните му елементи, както и интерфейсът към тъчскрий экрана, се намират в долната лява част на печатната платка. Важно е да се отбележи, че кварцът е поставен възможно най-близо до микроконтролера, като по този начин се гарантира доставянето на стабилен и надежден тактов сигнал на интегралната схема. Повечето от сигналните проводящи писти в тази част от платката пренасят нискоскоростни цифрови сигнали, поради което скъсяването на дължината им не е от първостепенно значение. Всички пинови съединители, които служат за включване на течнокристалния экран и за свързване на външни устройства към микроконтролера, са разположени в тази област от платката.

Съставните части на аналогия четиристъпален усилвател са разположени в горната дясна част на платката, в непосредствена близост до ЦАП-а на DDS системата. Всички проводящи писти, през които се пренася генерирания сигнал, са изключително къси. При поставяне на операционните усилватели AD8021 върху печатната платка са взети определени съображения, които са описани в каталожните данни⁽¹⁵⁾ на интегралната схема. Разположението на четирите операционни усилватели (**U16**, **U17**, **U19**, **U22**) наподобява примерното разположение, представено на фигура 62, извадена на фиг. А.8, от каталожните данни⁽¹⁵⁾ на AD8021. Компенсационните кондензатори са сложени в непосредствена близост до съответните им операционни усилватели. Производителят също препоръчва използването на предпазен пръстен, който представлява медна писта, свързана към изхода на усилвателя (6-ти пин) и ограждаща 5-ти пин и свързания към последния компенсационен кондензатор. Използването на предпазен пръстен се обосновява от високия импеданс на 5-ти пин на AD8021 и от ниския капацитет на компенсационния кондензатор. Пръстенът предпазва 5-ти пин от всякакъв шум, произведен от съседни електронни компоненти, и отстранява всякакви паразитни капацитети, които може да се добавят към капацитета на компенсационния кондензатор.

По принцип при свързване на електронните компоненти със земите на устройството (**AGND** и **DGND**) се използват широки и къси писти и големи проводящи проходи с цел постигане на нискоомна свързаност между заземлящите площи и електронните елементи. В противен случай токовете, които протичат през захранването на интегралните схеми, ще образуват пад на напрежение, изчислен по закона на Ом, с високоомните връзки. По този начин електрическият потенциал на общия проводник (заземлящите площи) на печатната платка ще се различава от идеалния нулев електрически потенциал. Ако разликата между реалния потенциал на земите и идеалния нулев потенциал е значителна, е възможно да се възпрепятства нормалното и правилното функциониране на принципната електрическа

схема. Все пак на платката са налични връзки към общия проводник, за които се използват тънки писти и малки проводящи проходи. Тези връзки се използват за установяване на ниско логичеки ниво на изводите на някои интегрални схеми и следователно през тях протичат незначителни токове.

В долната дясна част на платката е разположен интерфейсът към захранването. Пистите, които се свързват към захранващия слой, са широки и съответно нискоомни. Върху долната страна на печатната платка е закрепен линейния стабилизатор ADP7142 (**U6**), който се използва за получаването на нужното аналогово захранващо напрежение **+5VA** (спрямо **AGND**). Тъй като на входа на стабилизатора е подадено сравнително високото аналогово захранващо напрежение **+12VA**, а на изхода се очаква стабилизираното напрежение **+5VA**, падът на напрежение върху интегралната схема е сравнително голям (7V). Също така корпусът на схемата е с малки размери (TSOT-23-5). Поради тези причини производителят препоръчва⁽¹⁷⁾ използването на медни площи, свързани съответно към входа, изхода и заземяващия извод на ADP7142. Голямото количество мед разсейва топлината, произведена от линейния стабилизатор. Производителят препоръчва в каталожните данни⁽¹⁷⁾ използването на възможно най-голяма площ на медните площи, но също така споменава, че при добавянето на допълнителна медна площ след някаква определена стойност температурните характеристики на стабилизатора се подобряват незначително. За корпуса TSOT-23-5 производителят препоръчва добавянето на медна плоча с площ неповече от 500 mm² (виж таблица 6 от (17)). Площта на всяка една от използваните на проектираната платка медни площи е повече от 100 mm².

В принципната електрическа схема са използвани голям брой отделящи/изправящи кондензатори, които осигуряват нормалното функциониране на различните интегрални схеми. Ако разстоянието между захранващите изводи и кондензаторите е голямо, изправящото действие на последните се губи, поради което в захранването на интегралните схеми е възможно да се индуцират смущаващи напрежения, които да възпрепятстват нормалното и правилно функциониране на схемите. Затова проектантът се е постарал да постави тези отделящи кондензатори в непосредствена близост до съответните им интегрални схеми.

4.3.2 Захранващият слой (2-ри)

Този слой е представен на фиг. 4.4 и се използва за доставяне на необходимите захранващи напрежения на всички електронни компоненти.

И цифровите, и аналоговите захранващи напрежения се пренасят по платката, използвайки много широки проводящи писти, които не се вписват в нито един от „класовете“ писти от таблица 4.2. Изполвайки много широки и съответно нискоомни медни писти, захранващите напрежения може да се пренасасят на дълги разстояния от печатната платка. Ако пренасящите писти бяха тънки и съответно сравнително по-високоомни от използваните, върху медните писти би се отделил значителен пад на напрежение. Следователно различни интегрални схеми, които на принципната електрическа схема са свързани към еднакво захранващо напрежение, биха получили различни по стойност реални постояннотокови напрежения на захранващите си изводи.

Широките проводящи писти са разположени в участъци от платката, които са в близост до съответните интегрални схеми, които захранват. От широките медни писти се спускат по-тънки проводници, които се свързват към отделните електронни компоненти чрез проводящи проходи.

4.3.3 Заземяващият слой (3-ти)

Заземяващият слой е показан на фиг. 4.5. Всички използвани принципи за проектиране на заземяващия слой са описани в документа (5).

Вече беше коментирано няколкократно, че платката се захранва от две отделни, несвързани помежду си, захранвания - цифрово и аналогово. Захранващите напрежения, доставени от двете захранвания, се отмерват спрямо две отделни земи (**DGND** и **AGND**). За реализирането и на двете земи на печатната платка се използват две отделни заземяващи площи, които са свързани електрически точно и само в една единствена точка - звездната точка (от англ.: „star ground“). Тази обща точка представлява меден проводящ мост, свързващ двете заземяващи площи, на третия слой. Всички електрически потенциали, налични на платката, се измерват спрямо звездната точка, която има нулев електрически потенциал. Заземяващите площи не са свързани никъде другаде на платката и са изолирани с празнина на 3-тия слой. Също така цифровата схемотехника на платката е разположена върху цифровата земя (**DGND**), а аналоговата - върху аналоговата заземяваща плоча (**AGND**). По този начин обратният път на цифровите сигнални токове е ограничен физически само до цифровата заземяваща плоча (**DGND**), а обратният път на аналоговите сигнални токове минава физически само през аналоговата заземяваща плоча (**AGND**). Следователно тихите аналогови интегрални схеми са изолирани от шумната цифрова схемотехника, която в противен случай би смутила нормалната и прецизна дейност на аналоговата схемотехника.

4.4 Други особености на печатната платка

В тази точка са описани накратко други важни и забележими особености на проектираната печатна платка.

4.4.1 Размери

Печатната платка е квадратна и има физически размери 10cm × 10cm. Дебелината на платката зависи от производителя на печатната платка.

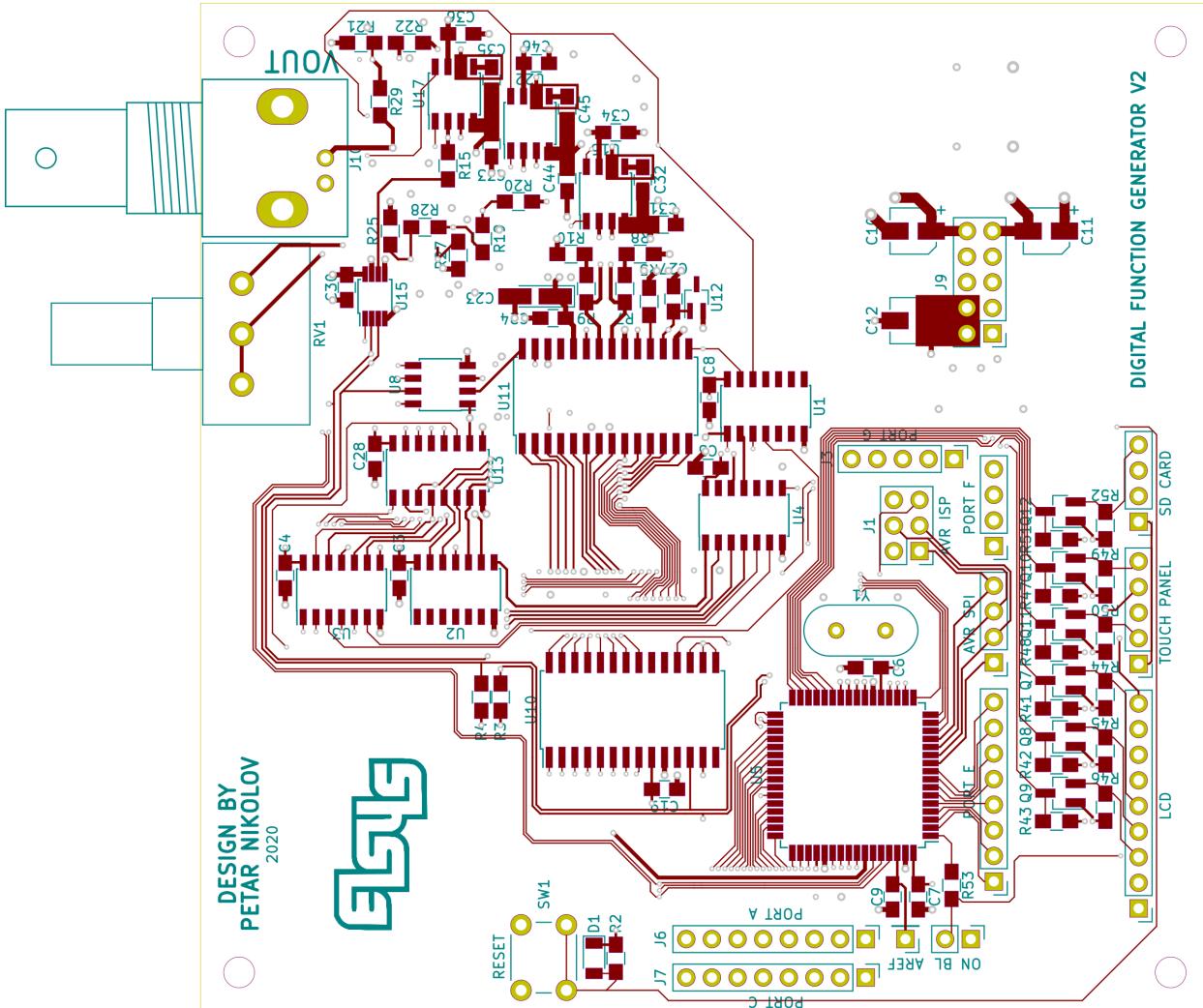
4.4.2 Монтиране

В четирите ъгли на платката са разположени дупки за стандартни болтове 3mm. Всички проводящи слоеве са прекъснати в областта около дупката за болтовете и следователно последните не влияят по никакъв начин на електрическата свързаност на платката.

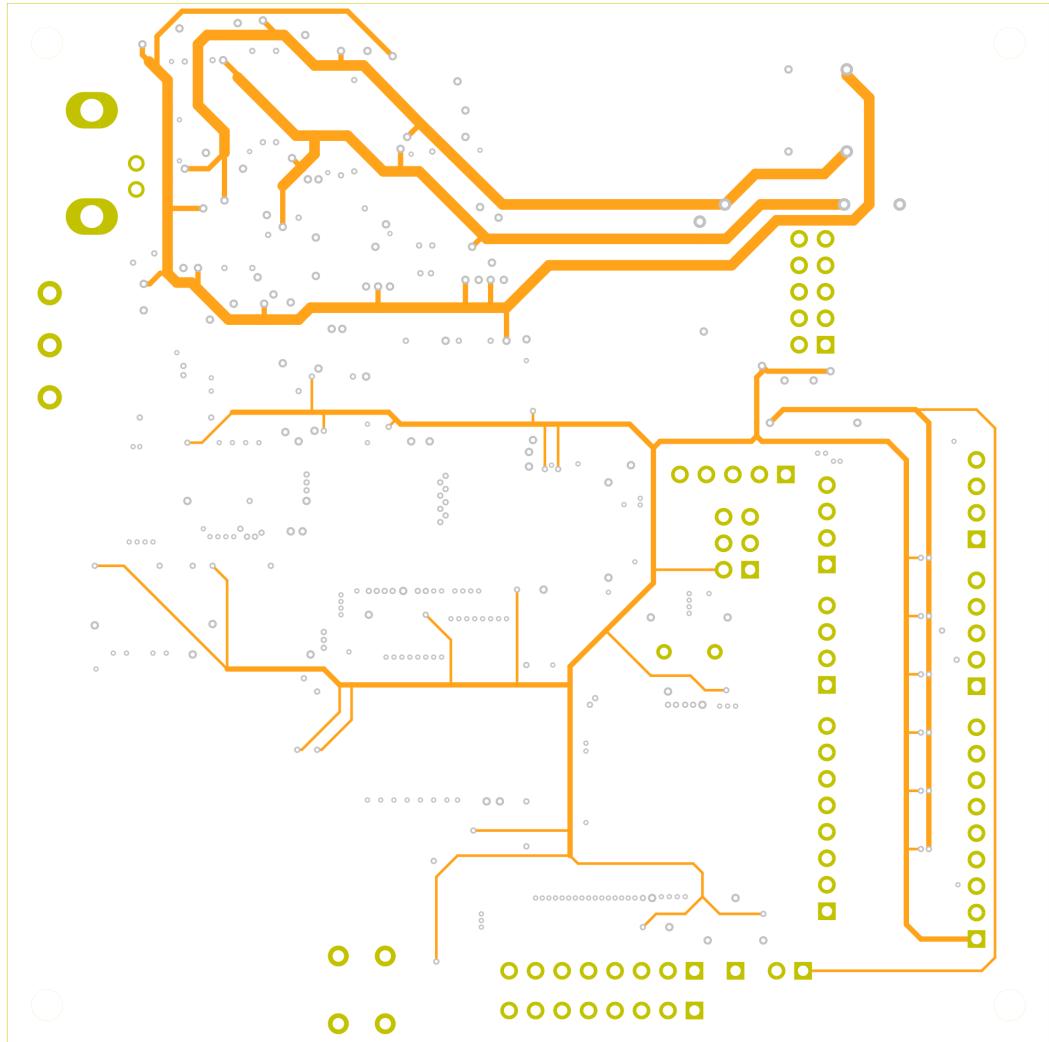
4.5 Визуализация на печатната платка с 3D модели

Проектираната печатна платка е визуализирана с помощта на 3D модели. Резултатите от визуализацията са представени на фиг. 4.7 - горната страна, и фиг. 4.8 - долната страна.

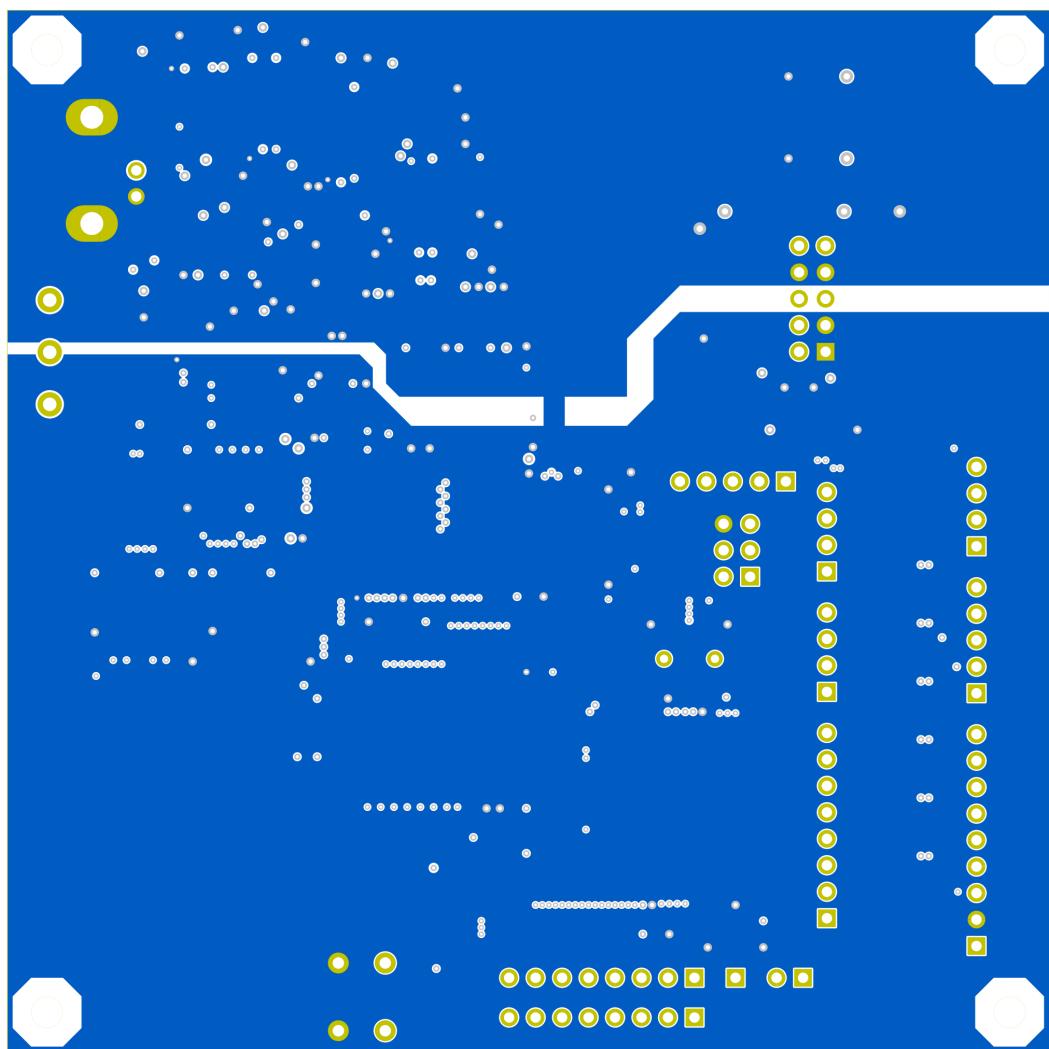
Повечето от използваните 3D модели са част от свободната вградена библиотека с 3D модели на KiCAD. Има само два 3D модела, които са взети от трети лица - моделите за BNC конектора и за механичния потенциометър. Уеб страниците на взетите модели може да се намерят съответно на (35) и (36).



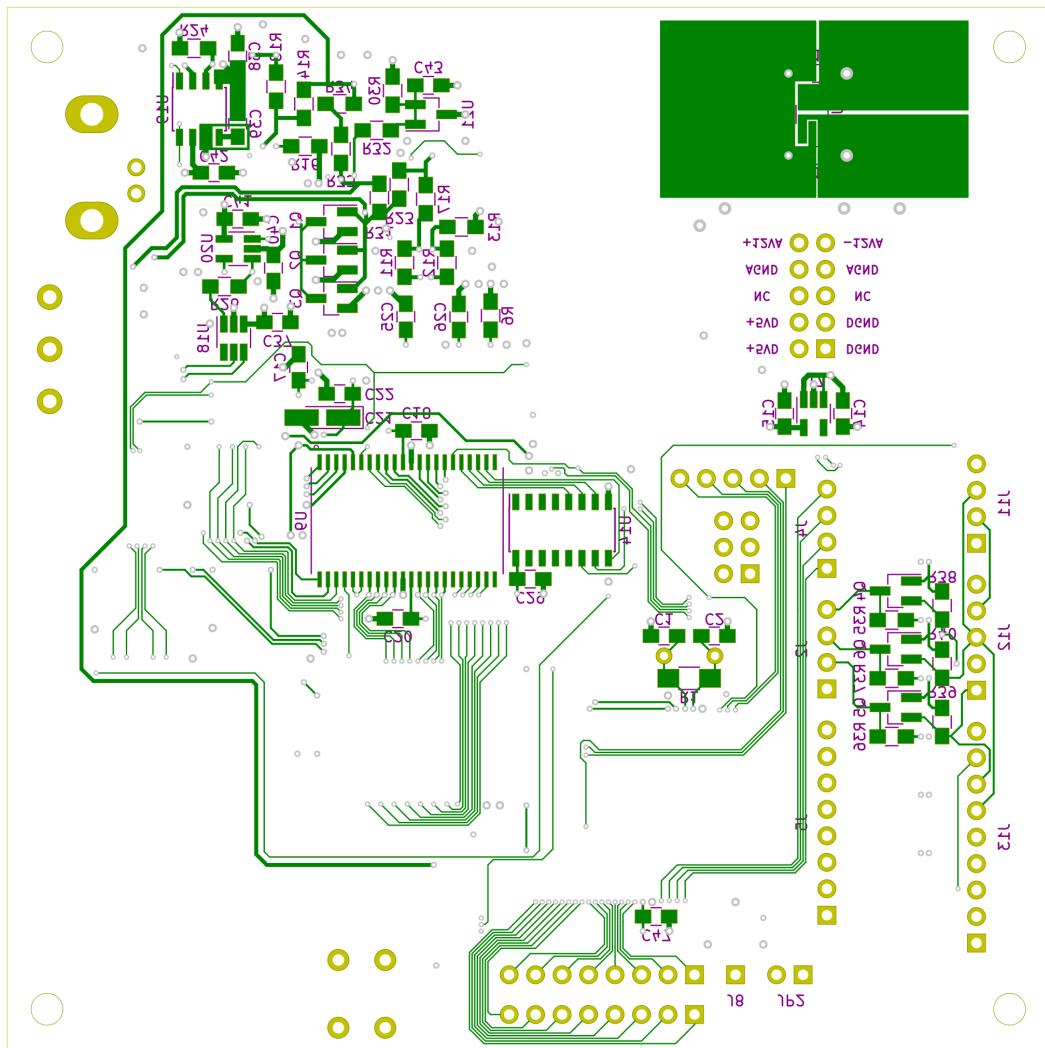
фиг. 4.3: 1-ви слой (Горен) на печатната платка



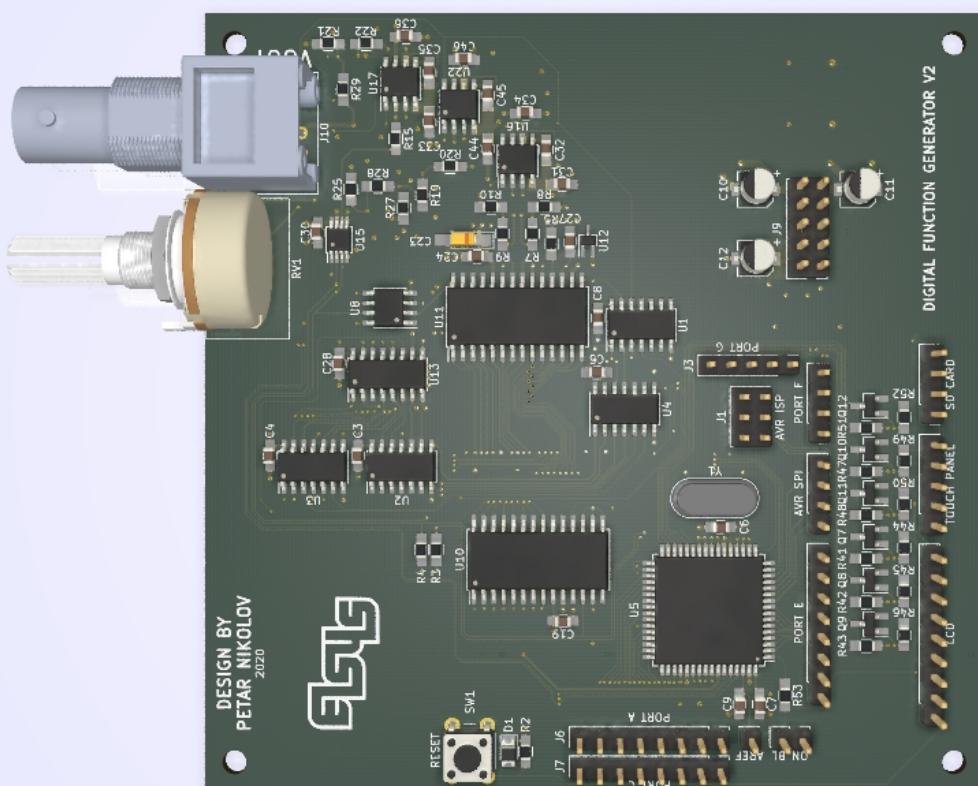
фиг. 4.4: 2-ри слой (Захранващ) на печатната платка



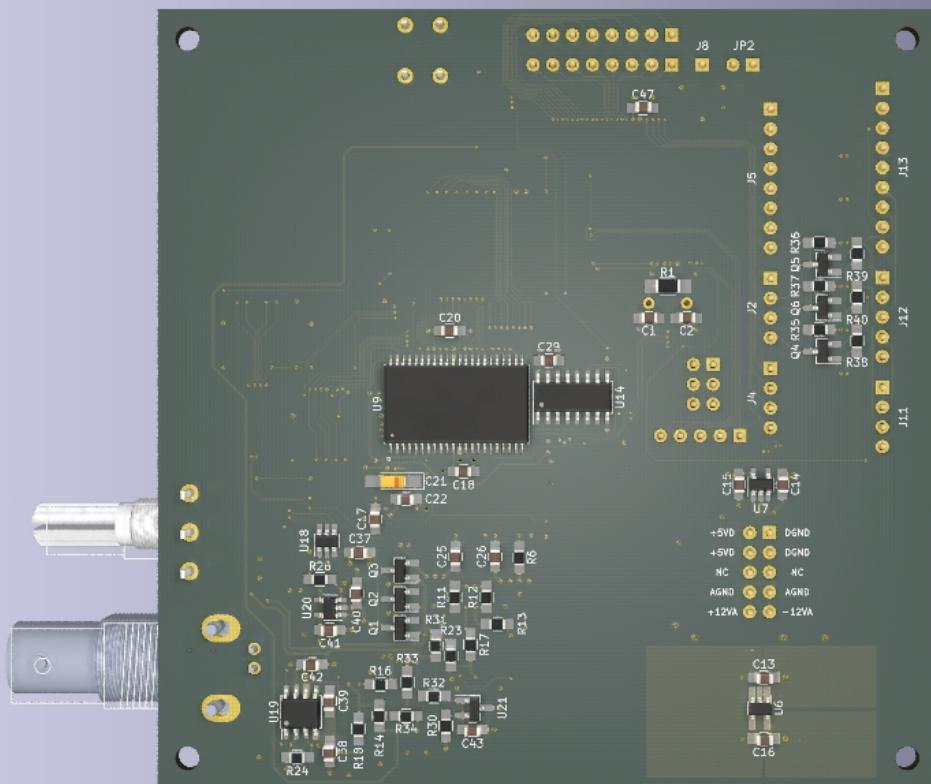
фиг. 4.5: 3-ти слой (Земя) на печатната платка



фиг. 4.6: 4-ти слой (Долен) на печатната платка



фиг. 4.7: 3D модел на печатната платка - горна страна



фиг. 4.8: 3D модел на печатната платка - добра страна

ГЛАВА V

Разработка на софтуерен драйвер за взаимодействие с тъчскрийн модула и алгоритъм за управление на устройството

5.0 Предпоставки

В тази глава се предполага, че читателят е запознат и с SPI комуникационния протокол, и с I²C комуникационния протокол⁽²³⁾. Затова в текста не се коментират спецификите на протоколите.

Също така се предполага, че читателят е запознат с принципите в програмирането на вградени микрокомпютърни системи и е запознат с езика AVR-C за програмиране на AVR микроконтролери.

5.1 Софтуерна среда за разработка на програмата на микроконтролера

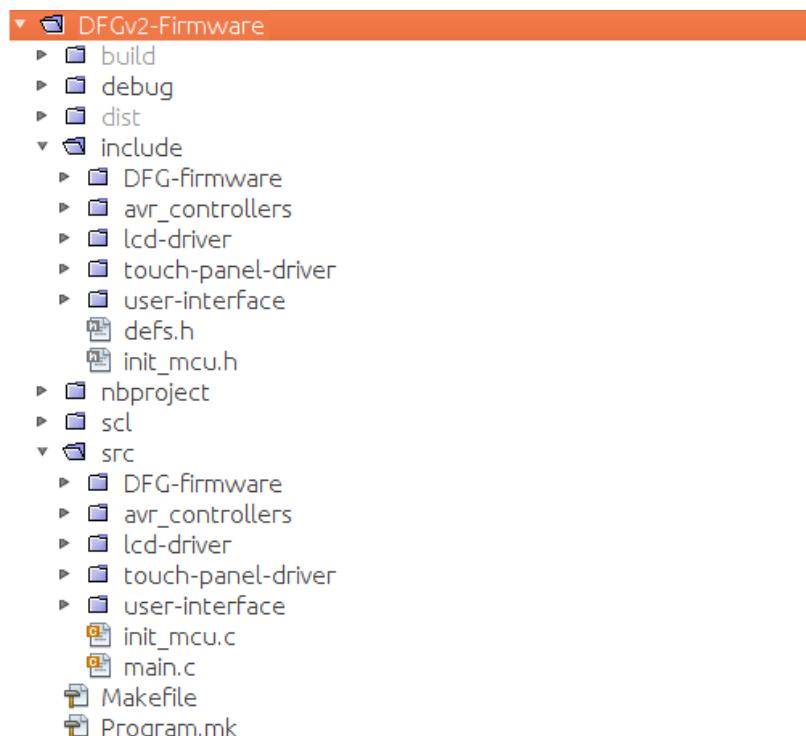
За разработка на управляващия софтуер е използван приложният програмен продукт „MPLAB X IDE v5.25“. Последният представлява интегрирана среда за разработка, удобна за създаване на управляващ софтуер за микроконтролерите, произвеждани от Microchip, включително и за използвания ATmega2561 микроконтролер. Продуктът се поддържа от споменатия производител на микроконтролерни системи, като повече информация за „MPLAB X IDE“ може да се намери на уебстраницата на Microchip за софтуерната среда⁽³⁷⁾.

Използваният приложен програмен продукт има изключително много функционалности, които улесняват работата при разработка на управляващия софтуер, спрямо обикновения текстов редактор. Някои от най-големите му предимства са:

- Автоматично поддържана файлова структура.
- Дебъгер, позволяващ изследването на разработения софтуер и влиянието му върху вътрешните памети на микроконтролера стъпка по стъпка.
- Възможност за разглеждане на записания в контролера Assembler код.
- Симулатор, който позволява емулирането както и на прости входно-изходни операции, така и на дейността на сложни периферни функционалности на микроконтролера като SPI, I²C интерфейсите.

5.2 Файлова структура на разработения софтуер

Управляващият софтуер е написан на езика за програмиране „AVR-C“. Последният представлява широкоразпространеният език за програмиране „C“ с допълнителни хедър файлове за управление на микроконтролерите с AVR структура, включително и ATmega2561. Използването на програмния език C предполага, че управляващият софтуер е съставен от съвкупност от хедър файлове (.h), имплементирани във файлове с програмен код (.c). На фиг. 5.1 е представена съкратената файлова структура на управляващия софтуер.



фиг. 5.1: Съкратена файлова структура на управляващия софтуер

Главната директория „DFGv2-Firmware“ съдържа всички съставни файлове, необходими за изграждането и компилирането на софтуерната програма на микроконтролера. Главната директория съдържа няколко поддиректории, които имат следните предназначения:

- „build/“: съдържа компилираните обектни файлове (.o) на управляващия софтуер. Тази директория се поддържа от MPLAB X и няма бъде коментирана в настоящата дипломна работа.
- „debug/“: съдържа компилираните обектни файлове (.o) на управляващия софтуер, удобни за изграждане на негова версия за дебъгване. Тази директория се поддържа от MPLAB X и няма бъде коментирана в настоящата дипломна работа.

- „*dist*/“: съдържа файлове, които се използват за програмиране на микроконтролера. Тази директория се поддържа от MPLAB X.
- „*nbproject*/“: конфигурационни файлове, използвани от MPLAB X. Тази директория няма да бъде коментирана в настоящата дипломна работа.
- „*include*/“: съдържа всичките интерфейсни хедър файлове (.h) на разработения софтуер.
- „*src*/“: съдържа всичките имплементационни файлове (.c) на разработения софтуер.
- „*scl*/“: съдържа SCL скриптове, които се използват при дебъгване от симулатора на MPLAB X. Тази директория няма да бъде коментирана в настоящата дипломна работа.

Както и интерфейсните файлове (.h), така и имплементационните файлове (.c) са подразделени в различни едноименни поддиректории съответно в „/include“ и в „/src“. Някои файлове не следват този разделителен принцип (например „/src/main.c“, който съдържа main функцията на програмата). Повечето интерфейсни хедър файлове (.h) декларират функционалности, които са имплементирани в едноименните им имплементационни файлове (.c). Има изключения към този организационен принцип (например файлът „/include/defs.h“). Различните поддиректории съдържат файлове, изпълняващи следните функционалности:

- „avr_controllers/“: реализира контролери за SPI и TWI (I^2C) интерфейсите на ATmega2561.
- „DFG-firmware/“: реализира алгоритъма за управление на устройството.
- „lcd-driver/“: реализира драйверен софтуер за управлението на драйверната интегралната схема (ILI9341) за течнокристалния екран.
- „touch-panel-driver/“: реализира драйверен софтуер за управлението на драйверната интегрална схема (XPT2046) за панела за допир.
- „user-interface/“: реализира примерен потребителски интерфейс.

Главната директория съдържа и два файла, използвани от програмата „make“ - главният „Makefile“, който е генериран от MPLAB X и се употребява за компилиране на програмата, и допълнителният файл „Program.mk“. Последният се използва за програмиране и на вградената програмна флеш памет, и на конфигурационните байтове (от англ.: „fuse bytes“) на ATmega2561.

5.3 Помощен код за реализиране на управляващата програма

Малка част от написания код е разработен с цел значително подобряване качеството на останалия програмен код или избягване на повтарящ се код. Описаните в тази точка функционалности и структури се използват в програмния код, реализиращ основните функционалности на управляващата програма.

5.3.1 Булеви променливи

Стандартният тип данни **bool** в C, който е дефиниран в „*<stdbool.h>*“, е удобен за ползване, но е неефикасен, защото все пак заема 1 байт памет. В една микроконтролерна система, при която наличните ресурси са ограничени (ATmega2561 има 8 KB SRAM памет), стандартният тип данни **bool** заема относително голямо количество памет, от която се използва само много малка част (1 бит от 1 байт).

По удачно решение е да се използва тип данни, при който всеки бит да представлява един флаг, т.е. едно булево състояние⁽³⁹⁾. Затова в разработения код се използва собствено дефинираният тип данни **fbool**, който всъщност представлява 8-битово целочислено число без знак (**uint8_t**). Следователно с една променлива от тип **fbool** може да се определят осем булеви състояния (флагове). Типът данни е дефиниран във файла „*include/defs.h*“.

За правилното използване на променлива от типа данни **fbool** е необходимо да се използват побитови операции. На фиг. 5.2 е представена дефиницията на типа, както и начинът му на употребяване.

```
#define FB00L0 0x01
#define FB00L1 0x02
#define FB00L2 0x04
#define FB00L3 0x08
#define FB00L4 0x10
#define FB00L5 0x20
#define FB00L6 0x40
#define FB00L7 0x80

typedef uint8_t fbool;

fbool flags;

// Отбележи 5-ти флаг (FB00L4) с истинност
flags |= FB00L4;

// Отбележи 2-ри флаг (FB00L1) с истинност
flags &= ~FB00L1;

// Провери състоянието на 1-ви (FB00L0) флаг
if (flags & FB00L0)
    // ако е истинност, изпълни "x"
else
    // ако е неистинност, изпълни "y"
```

фиг. 5.2: Използване на променлива от тип **fbool**

5.3.2 Операции с входно-изходни пинове

Понеже голям брой C функции на управляващия код трябва да манипулират състоянието на различни входно-изходни пинове на микроконтролера, е необходимо да се разработи начин за достъпване на пиновете на определени регистри със специално предназначение (SFR) на микроконтролера. Структурата **pin_ref_t** представлява своеобразна препратка към определен пин от определен входно-изходен порт. Самата структура от своя страна е създадена от два други собствено дефинирани типа данни - **SFR8_p**, който представлява указател към регистър със специално предназначение (SFR), и **SFRPIN**, показващ кой подред пин от използвания порт е указан със своеобразната препратка. Освен това за удобна употреба на структурата **pin_ref_t** са дефинирани макросите „*setpinref()*“, „*resetpinref()*“, които съответно променят стойността на „1“ или на „0“ на указания пин. От друга страна, макроса „*pinrefstate()*“ може да се използва за проверка състоянието на указания пин. Дефиницията на структурата **pin_ref_t**, както и макросите, използвани в нея, са извадени на фиг. 5.3, а на фиг. 5.4 са представени примери за използването на променливи от този тип.

```
...
35  typedef volatile uint8_t * SFR8_p;
36
37  typedef uint8_t SFRPIN;
..
41  struct pin_ref_t {
42      SFR8_p port;
43      SFRPIN pin;
44  };
45
46  #define setpinref(pin_ref) { \
47      *((pin_ref).port) |= _BV((pin_ref).pin); \
48  }
49
50  #define resetpinref(pin_ref) { \
51      *((pin_ref).port) &= ~_BV((pin_ref).pin); \
52  }
53
54  #define pinrefstate(pin_ref) (*pin_ref.port & _BV(pin_ref.pin))
```

фиг. 5.3: Дефиниция на структурата **pin_ref_t**

файл: „include/defs.h“

```

struct pin_ref_t pb4 = { &PORTB, PB4 };

// промени състоянието на пин PB4 на „1“
setpinref(pb4);

// промени състоянието на пин PB4 на „0“
resetpinref(pb4);

// провери състоянието на пин PB4
if (pinrefstate(pb4))
    // ако състоянието му е логическа „1“, направи „X“
else
    // ако състоянието му е логическа „0“, направи „Y“

```

фиг. 5.4: Примери за използване на променлива от тип **pin_ref_t**

На фиг. 5.5 е представен и друг полезен макрос „*setrst()*“ , който задава състоянието на определен бит от определен регистър със специално предназначение (SFR) според стойността на трета променлива. На същата фигура е показан и пример за употребяването на макроса, където става ясно, че е удобно да се използва в комбинация с параметър на функция от тип **fbool**.

```

#define setrst(reg, bit, to_set) { \
    if ((to_set)) (reg) |= (1 << (bit)); \
    else (reg) &= ~(1 << (bit)); \
}

void func(fbool parameter)
{
    setrst(PORTE, PE4, parameter);
}

```

фиг. 5.5: Дефиниция и пример за „*setrst()*“

5.3.3 SPI контролер

На фиг. 5.7 е представена структурата **spi_interface_t**, която се използва за инициализация на SPI интерфейса на микроконтролера. На фиг. 5.6 е представен имплементацията на набор от функции, които се използват за управление на SPI интерфейса на ATmega2561. Функцията „*mspi_init()*“ инициализира SPI интерфейса, като настройва сигналните си линии в съответния режим (входен или изходен) и включва SPI интерфейса в **Master** режим, което се постига чрез задаване на битовете **SPE** и **MSTR** на регистъра **SPCR**. Последният представлява контролен регистър за конфигуриране на SPI интерфейса на AVR устройствата. Функциите „*spi_set_speed()*“ и „*spi_set_data_mode()*“ служат съответно за задаване на скоростта на комуникация и за определяне на начина на пренос на данни. Последната функция във файла е „*spi_transfer()*“, с която се извършва едновременно пренос и прием на един байт данни. Преносът започва веднага след задаване стойността на регистъра

SPDR, след което микроконтролерът изчаква завършването на операцията. След успешен пренос приетите от отсрецната страна данни се записват автоматично от хардуерния SPI интерфейс в регистъра **SPDR**, чиято стойност се връща от функцията. Повече информация за специалните регистри **SPCR**, **SPSR** и **SPDR**, които настройват и управляват хардуерния SPI интерфейс на ATmega2561 може да се намери в каталожните му данни⁽⁷⁾ (виж стр. 197 ÷ 199).

```

1 void mspi_init(struct spi_interface_t *spi)
2 {
3     // configure SCK and MOSI pins as outputs
4     *spi->ddr_spi |= (1 << spi->mosi_pin) | (1 << spi->sck_pin);
5     // configure MISO pin as input
6     *spi->ddr_spi &= ~(1 << spi->miso_pin);
7
8     // enable SPI in Master mode;
9     SPCR = (1 << SPE) | (1 << MSTR);
10 }
11
12 void spi_set_speed(fbool clockrate)
13 {
14     /*
15      * flagsbool clockrate:
16      * bit 0: SPR0
17      * bit 1: SPR1
18      * bit 2: SPT2X
19      * bits 3-7: not used
20      */
21     setrst(SPCR, SPR0, clockrate & FB00L0);
22     setrst(SPCR, SPR1, clockrate & FB00L1);
23     setrst(SPSR, SPT2X, clockrate & FB00L2);
24 }
25
26 void spi_set_data_mode(fbool data_mode)
27 {
28     /*
29      * flagsbool data_mode:
30      * bit 0: CPHA
31      * bit 1: CPOL
32      * bit 2: DORD
33      * bits 3-7: not used
34      */
35     setrst(SPCR, CPHA, data_mode & FB00L0);
36     setrst(SPCR, CPOL, data_mode & FB00L1);
37     setrst(SPCR, DORD, data_mode & FB00L2);
38 }
39
40 uint8_t spi_transfer(uint8_t data)
41 {
42     // start SPI transmission
43     SPDR = data;
44     // wait for transmission to complete
45     while (!(SPSR & _BV(SPIF)));
46     // return received data from slave
47     return SPDR;
48 }
```

фиг. 5.6: Управление на SPI интерфейса на ATmega2561

файл: „src/avr_controllers/spi_controller.c“

```

23 struct spi_interface_t {
24     SFR8_p ddr_spi;
25     SFRPIN sck_pin;
26     SFRPIN miso_pin;
27     SFRPIN mosi_pin;
28 };

```

фиг. 5.7: Структурата **spi_interface_t**

файл: „include/avr_controllers/spi_controller.h“

5.3.4 TWI контролер

В проектираното устройство се използва I²C комуникация за настройване на определени интегрални схеми, изменяйки поведението на цифровия генератор на сигнали. AVR устройствата разполагат с т. нар. „Two Wire Interface“ (TWI), чиято единствена разлика с I²C интерфейс е в името. На фиг. 5.8 е показан интерфейсен код за управлението на споменатия TWI интерфейс. Деклариранияте функции имат следното предназначение:

- „twi_init()“: инициализира TWI интерфейса.
- „twi_set_speed()“: задава скоростта на TWI комуникацията.
- „twi_set_slave_address()“: задава TWI адреса на микроконтролера.
- „twi_start()“: сигнализира START условие.
- „twi_stop()“: сигнализира STOP условие.
- „twi_write_SLAW()“: изпраща SLA+W контролен байт по TWI интерфейса.
- „twi_write8()“: изпраща един байт за записване по TWI интерфейса.
- „twi_write()“: изпълнява една цялостна TWI комуникационна сесия, при която микроконтролерът изпраща за запис подадения масив в параметрите на функцията.

```

...
22 void twi_init(void);
23
24 void twi_set_speed(uint32_t, fbool);
25
26 void twi_set_slave_address(uint8_t);
27
28 uint8_t twi_start(void);
29
30 void twi_stop(void);
31
32 uint8_t twi_write_SLAW(uint8_t);
33
34 uint8_t twi_write8(uint8_t);
35
36 uint8_t twi_write(uint8_t, uint8_t *, size_t);
...

```

фиг. 5.8: Интерфейс за използване на TWI порта в програмния код

файл: „include/avr_controllers/twi_controllers.h“

На фиг. 5.9 е дадена част от управляващия код за TWI интерфейса. За изпращане на един байт се използва функцията „*twi_write8()*“, която първо зарежда байта данни в **TWDR** регистъра на микроконтролера. След това се задават битовете **TWINT** и **TWEN** на регистъра **TWCR**, с което се включва серийният интерфейс и започва предаването на данните. След завършването на предаването се проверява състоянието на TWI интерфейса и в случай, че отсрещната страна не е изпратила потвърждение на данните (ACK бит), функцията връща полученото състояние. В случай, че комуникацията е успешна, функцията връща стойността „0“. Функцията „*twi_write_SLAW()*“ (не е показана на фиг. 5.9) изпълнява същите операции като „*twi_write8()*“, но записва SLA+W в регистъра **TWDR**.

Изпращането на START („*twi_start()*“) или STOP („*twi_stop()*“) условие процедира по сходен начин, но тогава се задават съответно битовете **TWSTA** или **TWSTO** на **TWCR** вместо да се изпращат данни от **TWDR**.

Функцията „*twi_write()*“ обединява функционалността на всички гореспоменати функции, осъществявайки една цялостна TWI комуникационна сесия. Като параметри тази функция приема адреса на адресираното устройство и масив с байтове, които се изпращат последователно един след друг.

Повече детайли за TWI интерфейса и регистрите, които го настройват, може да се намерят в каталожните данни на ATmega2561⁽⁷⁾ (виж стр. 236 ÷ 264).

```

...
35 uint8_t twi_start(void)
36 {
37     // transmit START condition
38     TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN); // send START condition
39     while (!(TWCR & _BV(TWINT))); // wait for START condition to transmit
40     if (TW_STATUS != TW_START) // check status code for TWI
41         return TW_STATUS; // if start has not been transmitted, return error status
42     // TWCR &= ~_BV(TWEN); // manually clear TWEN bit of TWCR (for debugging)
43     return 0;
44 }
45
46 void twi_stop(void)
47 {
48     // transmit STOP condition
49     TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWSTO);
50 }
...
64 uint8_t twi_write8(uint8_t data)
65 {
66     // transmit a byte of data
67     TWDR = data; // load data into data register
68     TWCR = _BV(TWINT) | _BV(TWEN); // start data transmission
69     while (!(TWCR & _BV(TWINT))); // wait for data to transmit
70     if (TW_STATUS != TW_MT_DATA_ACK) // check status code for TWI
71         return TW_STATUS; // if ACK has not been received, return error status code
72     // TWCR &= ~_BV(TWEN); // manually clear TWEN bit of TWCR (for debugging)
73     return 0;
74 }
75
76 uint8_t twi_write(uint8_t slave, uint8_t *buffer, size_t size)
77 {
78     if (twi_start() != 0) return TW_STATUS; // transmit START condition
79     if (twi_write_SLAW(slave) != 0) return TW_STATUS; // transmit SLA+W
80     for (size_t i = 0; i < size; ++i) // transmit each data byte
81         if (twi_write8(buffer[i]) != 0) return TW_STATUS;
82     twi_stop(); // transmit STOP condition
83     return 0;
84 }

```

фиг. 5.9: Част от управляващия код за TWI интерфейса

файл: „src/avr_controllers/twi_controller.c“

5.4 Драйвер за управление на тъчскрийн модула

В състава на развойната платка на тъчскрийн модула са включени и драйверната интегрална схема за течнокристалния екран (ILI9341), и драйверната интегрална схема за панела за допир (XPT2046), като и двата компонента се управляват с SPI интерфейс. Следователно драйверният софтуер за управление на тъчскрийн модула следва да манипулира и SPI интерфейса, и състоянието на другите сигнални линии, участващи в комуникационния процес.

В процеса на разработка на управляващия софтуер изключително полезна референция е китайската библиотека, написана от LCDWIKI за управление на тъчскрийн модула с Arduino устройство. Структурана на библиотеката е показана на фиг. А.9, взета от ръководството на потребителя⁽⁴⁰⁾ за тъчскрийн модула. Китайският драйвер е съставен от три модула със следното предназначение:

- „LCDWIKI_SPI“: грижи се за хардуерната комуникация (изпращане и приемане на данни) с драйверния чип ILI9341.
- „LCDWIKI_GUI“: представлява просто API за рисуване на различни графики по LCD дисплея, като използва функционалността, реализирана в модула „LCDWIKI_SPI“.
- „LCDWIKI_TOUCH“: драйверен софтуер за интегралната схема, управляваща панела за допир (XPT2046).

В тази точка от 5-та глава се представят във фигури части от кода на китайската библиотека при обясняване на определени части от драйверния код, разработен от дипломанта. Важно е да се отбележи, че дипломантът не е копирал код от китайската библиотека. Последната е разработена за платформата Arduino, чийто код е несъвместим с използвания в дипломната работа програмен език „C“.

5.4.1 Комуникиране с драйверната интегрална схема за LCD екрана

Управлението на ILI9341 се постига чрез изпращане на определени команди, както и всякакви допълнителни параметри към изпратена команда ако има такива. Наборът от команди, който се използва от интегралната схема ILI9341 е изключително голям и е описан подробно в точка 8 от каталожните ѝ данни⁽²⁵⁾ (виж стр. 83 ÷ 201).

За подобряване съществено качеството на управляващия код за ILI9341 се използва едноименния хедър файл „*include/lcd-driver/chips/ILI9341.h*“. Съкратеният му вариант е представен на фиг. 5.10. В този файл са дефинирани макроси за резолюцията на екрана, всички команди за управление на ILI9341, както и няколко помощни макроси, употребявани

за удобство при изпращане на определени команди. Командите в хедър файла са разделени на три секции по подобие на каталожните данни⁽²⁵⁾ на ILI9341 с предназначение:

- Level 1 commands: команди, които се използват за манипулиране на течнокристалния екран. Чрез тях се осъществява рисуване върху экрана, промяна на яркостта му и т.н.
- Level 2 commands + Extended register command set: команди, с които се настройват хардуерните и електрическите параметри на интегралната схема. За да се позволи интерпретирането на тези команди, е необходимо да се подаде високо логическо ниво на извода EXTC на ILI9341. Тъй като не може да се провери състоянието на извода, дипломантът е установил опитно, че командите се изпълняват от схемата, и следователно целият набор от команди е позволен.

```

...
8 #ifndef ILI9341_H
9 #define ILI9341_H
10
11 #define LCD_RESX 240 // 0x00EF
12 #define LCD_RESY 320 // 0x013F
13
14 // Level 1 commands
15 #define CMD_NOP 0x00
...
64 // Level 2 commands
65 #define CMD_IFMODE 0xB0
...
94 // Extended register command set
95 #define CMD_POWER_CONTROL_A 0xCB
...
103 #define MADCTL_DEFAULT_VALUE 0x00
104 #define MADCTL_MY 0x80
105 #define MADCTL_MX 0x40
106 #define MADCTL_MV 0x20
107 #define MADCTL_ML 0x10
108 #define MADCTL_BGR 0x08
109 #define MADCTL_MH 0x04
110
111 #define DPI_16BIT 0x50
112 #define DPI_18BIT 0x60
113 #define DBI_16BIT 0x05
114 #define DBI_18BIT 0x06
115
116 #define BLCTRL_BCTRL 0x20
117 #define BLCTRL_DD 0x08
118 #define BLCTRL_BL 0x04
119
120 #ifdef __cplusplus
121 extern "C" {
122 #endif
123
124 #ifdef __cplusplus
125 }
126 #endif
127
128 #endif /* ILI9341_H */

```

фиг. 5.10: Наборът от команди на ILI9341

файл: „include/lcd-driver/chips/ILI9341.h“

На фиг. 5.11 е представена структурата **lcd_driver_t**, която се използва в управляващия код за осъществяване на комуникация с драйверната интегрална схема ILI9341. Структурата съдържа в себе си указател към структура **spi_interface_t**, с която се управлява SPI порта на микроконтролера, препратки (**pin_ref_t**) към 3-те допълнителни сигнални линии, текущото състояние на **MADCTL** регистъра на ILI9341 и хоризонталната и вертикалната резулюция на екрана. Предназначението на 3-те допълни сигнали линии е следното:

- **D/CX (dcx в кода)**: определя дали следващият изпратен байт е команда или параметър към командата. В състояние логическа „1“ ILI9341 приема изпратения байт като параметър, а в „0“ - като команда.
- **RESET (reset в кода)**: рестартира ILI9341 при подаване на високо логическо ниво.
- **CSX (csx в кода)**: при подаване на ниско логическо ниво драйверната схема участва в SPI комуникацията. В противен случай не обменя данни по SPI линиите.

```

...
19 // send a data byte via the D/CX line
20 #define _dcx_data(lcd_driver) { \
21     *lcd_driver->dcx.port |= (1 << lcd_driver->dcx.pin); \
22 }
23
24 // send a command byte via the D/CX line
25 #define _dcx_command(lcd_driver) { \
26     *lcd_driver->dcx.port &= ~(1 << lcd_driver->dcx.pin); \
27 }
28
29 // deselect the LCD driver (output a HIGH level to Chip Select)
30 #define _csx_high(lcd_driver) { \
31     *lcd_driver->csx.port |= (1 << lcd_driver->csx.pin); \
32 }
33
34 // select the LCD driver (output a LOW level to Chip Select)
35 #define _csx_low(lcd_driver) { \
36     *lcd_driver->csx.port &= ~(1 << lcd_driver->csx.pin); \
37 }
38
39 // output a HIGH level to the RESET line
40 #define _reset_high(lcd_driver) { \
41     *lcd_driver->reset.port |= (1 << lcd_driver->reset.pin); \
42 }
43
44 // output a LOW level to the RESET line
45 #define _reset_low(lcd_driver) { \
46     *lcd_driver->reset.port &= ~(1 << lcd_driver->reset.pin); \
47 }
...
64 struct lcd_driver_t {
65     struct spi_interface_t *spi; // SPI interface
66     struct pin_ref_t dcx; // Data/Command
67     struct pin_ref_t reset; // Reset
68     struct pin_ref_t csx; // Chip Select
69     uint8_t madctl;
70     uint16_t res_x;
71     uint16_t res_y;
72 };
73
74 void lcd_driver_init(struct lcd_driver_t *, fbool);

```

фиг. 5.11: Структурата, с която се реализира програмния код за комуникация с ILI9341

файл: „include/lcd-driver/lcd-driver.h“

За всеки от допълнителните сигнални линии, необходими за правилна комуникация с драйверната интегрална схема, са дефинирани по една двойка макроси, показани на фиг. 5.11, като единият подава високо логическо ниво на съответната сигнална линия, а другият - ниско логическо ниво. Тези макроси подобряват качеството на управляващия код за ILI9341, като не е препоръчително да се използват извън рамките на имплементационния код.

Функцията „*lcd_driver_init()*“ инициализира променлива от тип **lcd_driver_t**. Имплементацията ѝ е дадена на фиг. 5.12. Според подадения аргумент **options** функцията може да инициализира SPI порта. След това запаметява стойността по подразбиране на регистъра **MADCTL** при стартиране на чипа ILI9341. Според стойността на **MADCTL** се определя хоризонталната и вертикалната резолюция на екрана. След това допълнителните сигнални линии за комуникация с ILI9341 се настройват в неутрално състояние.

```

20 void lcd_driver_init(struct lcd_driver_t *driver, fbool options)
21 {
22     /* fbool options:
23      * bit 0: initialize Master SPI
24      * bits 1-7: not used
25      */
26     // optionally initialize the SPI interface
27     if (options & FBOOL0)
28         mspi_init(driver->spi);
29
30     driver->madctl = MADCTL_DEFAULT_VALUE;
31     driver->res_x = (driver->madctl & MADCTL_MV) ? LCD_RESY : LCD_RESX;
32     driver->res_y = (driver->madctl & MADCTL_MV) ? LCD_RESX : LCD_RESY;
33     _dcx_data(driver);
34     _csx_high(driver);
35     _reset_high(driver);
36 }
```

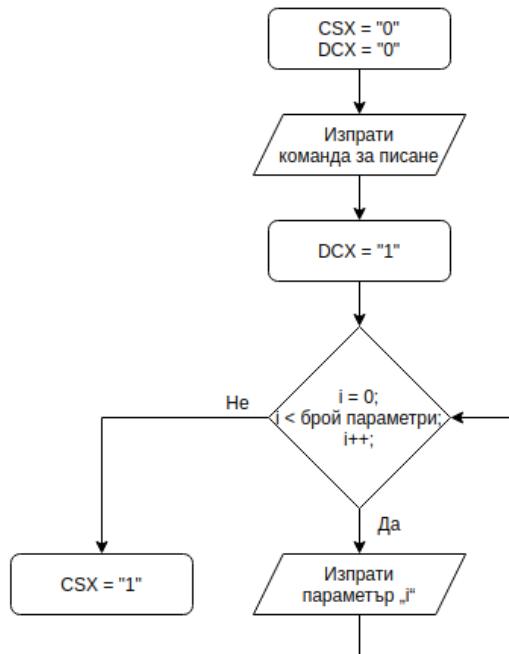
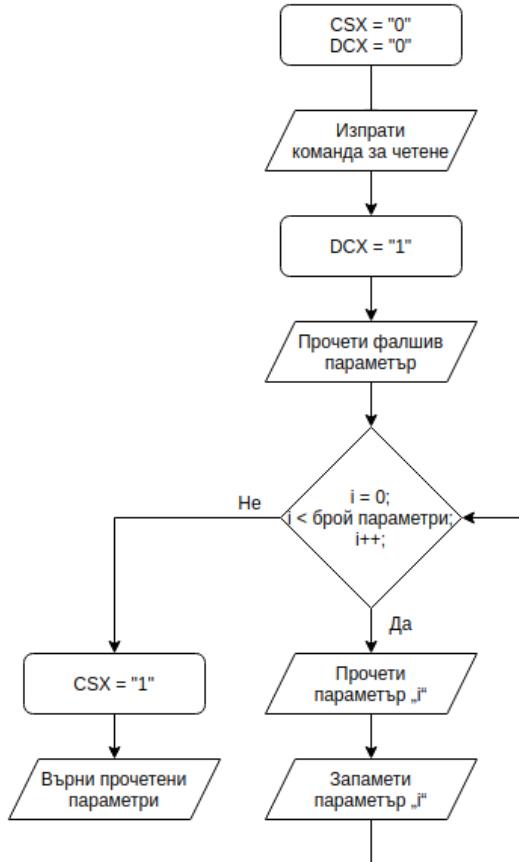
фиг. 5.12: Инициализация на структурата **lcd_driver_t**

файл: „src/lcd-driver/lcd-driver.c“

MADCTL е вграден в драйверната интегрална схема ILI9341 8-битов регистър, който определя подхода за достъп до вградената графична памет (GRAM) на чипа. На фиг. A.10 е изкарана таблица от каталожните данни⁽²⁵⁾ на ILI9341, която описва предназначението на битовете на **MADCTL**:

- **MY**: определя посоката на достъп до редовете (горе-долу или долу-горе).
- **MX**: определя посоката на достъп до колоните (ляво-дясно или дясно-ляво).
- **MV**: разменя редовете с колоните и обратното.
- **ML**: определя вертикалната посока на опресняване (горе-долу или долу-горе).
- **BGR**: определя начинът на интерпретиране на една клетка от графичната памет (RGB или BGR).
- **MH**: определя хоризонталната посока на опресняване (ляво-дясно или дясно-ляво).

Следователно ако битът **MV** на **MADCTL** е зададен, то е необходимо да се разменят стойностите на хоризонталната и вертикалната резултация, както е показано на редовете 31 ÷ 32 от фиг. 5.12.



Чипът **ILI9341** приема два типа команди от отсрецната страна - за прочитане на информация от чипа или за записване на информация в чипа.

На фиг. 5.13 е дадена блокова схема, показваща обработването на команда за четене от **ILI9341**. Първоначално микроконтролерът избира чипа да участва в SPI комуникацията (**CSX** = „0“), след което изпраща командата (**DCX** = „0“). После микроконтролерът се приготвя да приеме данни (**DCX** = „1“). При използването на която и да е команда за четене от **ILI9341** чипът винаги изпраща фалшив и невалиден първи параметър, който трябва да се игнорира от микроконтролера. След това започва последователният прием на данни параметър по параметър, байт по байт. Прочитайки всички нужни параметри, микроконтролера откача чипа от SPI интерфейса (**CSX** = „1“) и управляващият код връща прочетените параметри за по-нататъчна обработка.

От друга страна, на фиг. 5.14 е представена блокова схема, показваща обработка на команда за запис в ILI9341. Подходът за обработка на команда за запис е аналогична с подхода за обработка на команда за четене на фиг. 5.13. Единствената разлика е, че при подадена команда за запис микроконтролера изпраща параметри, допълващи командата, вместо да приема данни, изпратени от драйверната интегрална схема.

```

569 static void _lcd_command_no_parameter(struct lcd_driver_t *driver,
570     uint8_t command)
571 {
572     _csx_low(driver); // select the device
573     _dcx_command(driver); // transmit a command
574     spi_transfer(command); // transmit the command
575     _csx_high(driver); // deselect the device
576 }
577
578 static uint8_t _lcd_command_read_single_byte(struct lcd_driver_t *driver,
579     uint8_t command)
580 {
581     uint8_t result;
582
583     _dcx_command(driver); // transmit a command
584     _csx_low(driver); // select the device
585     spi_transfer(command); // transmit the command
586     _dcx_data(driver); // transmit data
587     spi_transfer(_LCD_DRIVER_NO_VALUE); // read dummy parameter (1st)
588     result = spi_transfer(_LCD_DRIVER_NO_VALUE); // read data (2nd)
589     _csx_high(driver); // deselect the device
590     return result;
591 }
592
593 static void _lcd_command_read_data(struct lcd_driver_t *driver,
594     uint8_t command, uint8_t *data, short nparams)
595 {
596     // WARNING: the 'data' array must be with length 'nparams'
597     _dcx_command(driver); // transmit a command
598     _csx_low(driver); // select the device
599     spi_transfer(command); // transmit the command
600     _dcx_data(driver); // transmit data
601     spi_transfer(_LCD_DRIVER_NO_VALUE); // read dummy parameter (1st)
602     for (short i = 0; i < nparams; ++i)
603         data[i] = spi_transfer(_LCD_DRIVER_NO_VALUE); // receive each data byte
604     _csx_high(driver); // deselect the device
605 }
606
607 static void _lcd_write_command(struct lcd_driver_t *driver,
608     uint8_t command, uint8_t *data, short nparams)
609 {
610     // WARNING: the 'data' array must be with length 'nparams'
611     _dcx_command(driver); // transmit a command
612     _csx_low(driver); // select the device
613     spi_transfer(command); // transmit the command
614     _dcx_data(driver); // transmit data
615     for (short i = 0; i < nparams; ++i)
616         spi_transfer(data[i]);
617     _csx_high(driver); // deselect the device
618 }
```

фиг. 5.15: Статични функции, реализиращи процеса на изпращане на команда към ILI9341

файл: „src/lcd-driver/lcd-driver.c“

На фиг. 5.15 е представена реализацията на изложените блокови схеми в програмен код. Показани са четири функции, които са дефинирани статично във файла „src/lcd-driver/lcd-driver.c“, т.е. те не са достъпни извън файла, в който са дефинирани, и спомагат за имплементацията на другите функции в същия файл. Предназначението на четирите, статично дефинирани във файла, функции е:

- „_lcd_command_no_parameter()“: изпраща команда за записване без никакви допълнителни параметри. Примери за други функции, имплементирани с тази статична функция, са показани на фиг. 5.16.

```

82 void lcd_nop(struct lcd_driver_t *driver)
83 {
84     _lcd_command_no_parameter(driver, CMD_NOP);
85 }
86
87 void lcd_swreset(struct lcd_driver_t *driver)
88 {
89     _lcd_command_no_parameter(driver, CMD_SWRESET);
90 }
```

фиг. 5.16: Функциите „lcd_nop()“ и „lcd_swreset()“

файл: „src/lcd-driver/lcd-driver.c“

- „_lcd_command_read_data()“: изпраща команда за четене, записвайки всички получени параметри в масив от данни, който трябва да е с достатъчно голям размер. На фиг. 5.17 е дадена функцията „lcd_read_display_status()“, която изпраща командата **RDDST** (09h)⁽²⁵⁾ и прочита 32-битов параметър, върнат от драйверния чип.

```

92 uint32_t lcd_read_display_status(struct lcd_driver_t *driver)
93 {
94     uint8_t data[4];
95     uint32_t result;
96
97     _lcd_command_read_data(driver, CMD_RDDST, data, 4);
98     result = ((uint32_t)(data[0])
99                 + ((uint32_t)(data[1]) << 8)
100                + ((uint32_t)(data[2]) << 16)
101               + ((uint32_t)(data[3]) << 24));
102
103 }
```

фиг. 5.17: Функцията „lcd_read_display_status()“

файл: „src/lcd-driver/lcd-driver.c“

- „_lcd_command_read_single_byte()“: предназначена е да се използва с команди за четене, които връщат само един параметър. Приложение на тази статична функция е дадено на фиг. 5.18.

```

110 uint8_t lcd_read_display_MADCTL(struct lcd_driver_t *driver)
111 {
112     driver->madctl = _lcd_command_read_single_byte(driver, CMD_RDDMADCTL);
113     return driver->madctl;
114 }

```

фиг. 5.18: Функцията „*lcd_read_display_MADCTL()*“

файл: „src/lcd-driver/lcd-driver.c“

- „*_lcd_write_command()*“: изпраща команда за писане заедно с всичките ѝ необходими допълнителни параметри. На фиг. 5.19 е представена функцията „*lcd_memory_access_control()*“, която използва командата за запис **MADCTL** (36h)⁽²⁵⁾

```

297 void lcd_memory_access_control(struct lcd_driver_t *driver, uint8_t madctl)
298 {
299     driver->madctl = madctl;
300     driver->res_x = (madctl & MADCTL_MV) ? LCD_RESY : LCD_RESX;
301     driver->res_y = (madctl & MADCTL_MV) ? LCD_RESX : LCD_RESY;
302     _lcd_write_command(driver, CMD_MADCTL, &madctl, 1);
303 }

```

фиг. 5.19: Функцията „*lcd_memory_access_control()*“

файл: „src/lcd-driver/lcd-driver.c“

Програмният код на всяка функция от управляващия софтуер, имплементираща някоя от командите на ILI9341, включва извикване на една от четирите статични функции, коментирани досега. Разликите между различните функции се изразяват в начина на оформление на изпратените параметри (при команди за запис) или начина на обработка на получените данни (при команди за четене). На фиг. 5.20 е представена функцията „*lcd_pixel_format_set()*“, която изпраща командата за запис **PIXSET** (3Ah)⁽²⁵⁾, задаваща формата за цвета на всеки пиксел (16 бита/пиксел или 18 бита/пиксел). Сравнявайки функциите, показани на фиг. 5.19 и фиг. 5.20, се забелязва, че и двете използват статичната функция „*_lcd_write_command()*“ в своя имплементационен код. Разликата е, че функцията „*lcd_memory_access_control()*“ използва подадения аргумент към параметъра **madctl** за манипулиране на структурата **driver**, докато функцията „*lcd_pixel_format_set()*“ единствено изпраща командата за запис без да извършва допълнителни операции.

```

326 void lcd_pixel_format_set(struct lcd_driver_t *driver, uint8_t format)
327 {
328     // NOTE: please use the DPI_xBIT/DBI_xBIT macros in the chip's header file
329     _lcd_write_command(driver, CMD_PIXSET, &format, 1);
330 }

```

фиг. 5.20: Функцията „*lcd_pixel_format_set()*“

файл: „src/lcd-driver/lcd-driver.c“

След захранване на течнокристалния екран и драйверната схема (ILI9341), която го управлява, е наложително да се настрои екрана преди да се използва. На фиг. 5.21 е показана последователност от операции (извикване на функции), които трябва да се извършат преди изпращането на каквито и да са други команди за манипулация на екрана. Първо се извиква функцията „*lcd_reset()*“, която извършва хардуерно презареждане на драйверния чип. Имплементацията на спомената функция е представена на фиг. 5.22. Първо микроконтролерът развързва драйверния чип от SPI комуникационните линии (ред 74-ти), след което изпраща импулс по сигналната линия **RESET**. Необходимият електрически импулс за презареждане на драйверния чип ILI9341 е описан в каталожните му данни⁽²⁵⁾ (т. 15.4). На фиг. A.11 е показана времевата диаграма, описваща процеса на презареждане, от каталожните данни⁽²⁵⁾ на ILI9341.

```

#define RESET_DELAY 120

// Последователност от операции за включване на екрана
lcd_reset(&driver);
_delay_ms(RESET_DELAY);
lcd_power_on(&driver);

```

фиг. 5.21: Последователност от операции за инициализация на екрана

```

72 void lcd_reset(struct lcd_driver_t *driver)
73 {
74     _csx_high(driver);
75     _reset_low(driver);
76     _delay_us(20);
77     _reset_high(driver);
78 }

```

фиг. 5.22: Хардуерно презареждане на ILI9341

файл: „src/lcd-driver/lcd-driver.c“

След успешно презареждане на екрана се извиква функцията „*lcd_power_on()*“, която извършва необходимото настройване на драйверния чип. Програмният код на функцията е даден на фиг. 5.23. Тя е съставена от поредица от команди, повечето от които настройват

електрически или други хардуерни параметри на драйверния чип. Последователността от команди е взета от библиотеката „LCDWIKI_SPI“ на LCDWIKI, като на фиг. A.12 е даден откъс от програмния код на библиотеката, показващ споменатата поредица. Използваните команди са детайлно описани в каталожните данни⁽²⁵⁾ на ILI9341.

```

38 void lcd_power_on(struct lcd_driver_t *driver)
39 {
40     uint8_t pgc_values[15] = { 0x0f, 0x26, 0x24, 0xb, 0xe, 0x9, 0x54, 0xa8, 0x46, 0xc, 0x17, 0x9, 0
41     uint8_t ngc_values[15] = { 0x00, 0x19, 0x1b, 0x4, 0x10, 0x7, 0x2a, 0x47, 0x39, 0x3, 0x6, 0x6, 0
42
43     lcd_swreset(driver);
44     _delay_ms(10);
45     lcd_display_off(driver);
46     lcd_interface_control(driver, 0x01, 0x01, 0x00);
47     lcd_power_control_b(driver, 0x81, 0x30);
48     lcd_power_on_sequence_control(driver, 0x64, 0x03, 0x12, 0x81);
49     lcd_driver_timing_control_a(driver, 0x85, 0x10, 0x78);
50     lcd_power_control_a(driver, 0x34, 0x02);
51     lcd_pump_ratio_control(driver, 0x20);
52     lcd_driver_timing_control_b(driver, 0x00);
53     lcd_rgb_interface_signal_control(driver, 0x00);
54     lcd_display_inversion_control(driver, 0x00);
55     lcd_power_control_1(driver, 0x21);
56     lcd_power_control_2(driver, 0x11);
57     lcd_vcom_control_1(driver, 0x3F, 0x3C);
58     lcd_vcom_control_2(driver, 0xB5);
59     // lcd_memory_access_control(driver, MADCTL_MY | MADCTL_BGR);
60     lcd_pixel_format_set(driver, DPI_16BIT | DBI_16BIT);
61     lcd_frame_rate_control_1(driver, 0x00, 0x1B);
62     lcd_enable_3G(driver, 0x00);
63     lcd_gamma_set(driver, 0x01);
64     lcd_positive_gamma_correction(driver, pgc_values);
65     lcd_negative_gamma_correction(driver, ngc_values);
66     lcd_entry_mode_set(driver, 0x07);
67     lcd_sleep_out(driver);
68     _delay_ms(150);
69     lcd_display_on(driver);
70 }
```

фиг. 5.23: Инициализация на ILI9341

файл: „src/lcd-driver/lcd-driver.c“

5.4.2 Рисуване върху LCD экрана

След като се захрани и инициализира драйверният чип ILI9341, последният във всеки един момент отрязва съдържанието на вградената си графична памет (GRAM) върху течнокристалния экран. Следователно непосредствено след успешно записване на нова стойност в дадена клетка от паметта състоянието (цветът) на съответния на клетката пиксел от LCD экрана се променя.

На фиг. A.13 е изобразена структурата на графичната памет (GRAM), от каталожните данни⁽²⁵⁾ на ILI9341. На фигурата са показани два адресни брояча - „Column Counter“ и „Page Counter“. Първият показва колоната на адресираната клетка, а другият - реда на клетката. Комбинацията от стойностите на двата брояча винаги определя и посочва само една единствена клетка в графичната памет.

Командата **CASET**⁽²⁵⁾ (2Ah) се използва за дефиниране на прозорец от колони, които микроконтролерът може да достъпи и манипулира. Командата изисква два 16-битови аргумента - началната колона (**SC**) и крайната колона (**EC**), като и двете са включени в дефинирания прозорец. Всякакъв опит за записване на стойност в клетки с колони, извън дефинирания обхват, се игнорира от драйверната интегрална схема. Функцията „*lcd_column_address_set()*“, чиято имплементация е показана на фиг. 5.24, се използва в управляващия софтуер за дефинирането на прозорец от достъпни колони. Освен че правилно форматира подадените 16-битови аргументи за изпращане, функцията също така извършва проверки за подаване на невалидни аргументи, като връща ненулева стойност при получен невалиден аргумент.

Командата **PASET**⁽²⁵⁾ (2Bh) дефинира прозорец от редове, които микроконтролерът може да достъпи, и е аналогична на командата **CASET**. Функцията, използваща **PASET**, е „*lcd_page_address_set()*“ и е аналогична на функцията „*lcd_column_address_set()*“.

```

189 signed short lcd_column_address_set(struct lcd_driver_t *driver,
190         uint16_t sc, uint16_t ec)
191 {
192     uint8_t data[4];
193
194     // SC[15:0] <= EC[15:0]
195     if (sc > ec) return -1;
196     // SC and EC cannot be out of range
197     if (sc > driver->res_x || ec > driver->res_x) return -2;
198
199     data[0] = sc >> 8;      // 1st parameter
200     data[1] = sc & 0xFF;    // 2nd parameter
201     data[2] = ec >> 8;    // 3rd parameter
202     data[3] = ec & 0xFF;   // 4th parameter
203     _lcd_write_command(driver, CMD_CASET, data, 4);
204     return 0;
205 }
206
207 signed short lcd_page_address_set(struct lcd_driver_t *driver,
208         uint16_t sp, uint16_t ep)
209 {
210     uint8_t data[4];
211
212     // SP[15:0] <= EP[15:0]
213     if (sp > ep) return -1;
214     // SP and EP cannot be out of range
215     if (sp > driver->res_y || ep > driver->res_y) return -2;
216
217     data[0] = sp >> 8;      // 1st parameter
218     data[1] = sp & 0xFF;    // 2nd parameter
219     data[2] = ep >> 8;    // 3rd parameter
220     data[3] = ep & 0xFF;   // 4th parameter
221     _lcd_write_command(driver, CMD_PASET, data, 4);
222     return 0;
223 }
```

фиг. 5.24: Задаване на достъпен за манипулиране прозорец от графичната памет на ILI9341

файл: „src/lcd-driver/lcd-driver.c“

Записването на данни в графичната памет на ILI9341 се осъществява с команда **RAMWR**⁽²⁵⁾ (2Ch). Тя приема неограничен брой параметри, като всеки един от тях представлява RGB цвят. Поведението на тази команда е следното:

- 1) Адресните броячи започват да сочат към началната колона (**SC**) и ред (**SP**), зададени съответно от командите **CASET** и **PASET**.
- 2) Стойността на първия получен параметър се записва в първата клетка.
- 3) Адресният брояч на колони минава на следващата поред колона, т.е. стойността му се увеличава с единица.
- 4) Продължава последователно да се записват стойностите на параметрите в клетките от графичната памет на драйверния чип.
- 5) Ако адресният брояч на колони достигне своята крайна стойност (**EC**), той се връща към началната колона (**SC**) и адресният брояч на редове минава на следващия ред.
- 6) Записването на стойности в графичната памет приключва, когато микроконтролерът спре да изпраща допълнителни параметри към командата.

На фиг. 5.25 е представена имплементацията на функцията „*lcd_memory_write()*“.

Последната приема като аргумент масив от цветове (16-битови числа), които са изпратени като параметри на командата **RAMWR**.

```

225 void lcd_memory_write(struct lcd_driver_t *driver, uint16_t *data, uint32_t size)
226 {
227     _dcx_command(driver); // transmit a command
228     _csx_low(driver); // select the device
229     spi_transfer(CMD_RAMWR); // transmit the command
230     _dcx_data(driver); // transmit data
231     for (uint32_t i = 0; i < size; ++i) {
232         spi_transfer(data[i] >> 8);
233         spi_transfer(data[i] & 0xFF);
234     }
235     _csx_high(driver); // deselect the device
236 }
237
238 void lcd_memory_write_optimized(struct lcd_driver_t *driver, uint16_t color, uint32_t size)
239 {
240     // NOTE: for transmitting only one color
241     _dcx_command(driver); // transmit a command
242     _csx_low(driver); // select the device
243     spi_transfer(CMD_RAMWR); // transmit the command
244     _dcx_data(driver); // transmit data
245     for (uint32_t i = 0; i < size; ++i) {
246         spi_transfer(color >> 8);
247         spi_transfer(color & 0xFF);
248     }
249     _csx_high(driver); // deselect the device
250 }
```

фиг. 5.25: Последователно записване на данни в графичната памет на ILI9341

файл: „src/lcd-driver/lcd-driver.c“

Функцията „*lcd_memory_write()*“ има един съществен недостатък: всяка стойност, изпратена като допълнителен параметър на **RAMWR**, изисква един отделен елемент на масива **data**. За изрисуване на големи площи от течнокристалния еcran функцията би се нуждаела от масив с изключително голям размер, който би запълнил голяма част от малко количество вградена памет на микроконтролера. Също на фиг. 5.25 е представена и функцията „*lcd_memory_write_optimized()*“. Тя изпълнява абсолютно същите операции като функцията „*lcd_memory_write()*“, но вместо да използва масив от цветове, приема само един 16-битов цвят **color**. Предимството на оптимизираната функция за запис е, че може да запълни голяма част от графичната памет на драйверната интегрална схема, заемайки незначително количество от вградената памет на микроконтролера. Недостатъка, разбира се, на „*lcd_memory_write_optimized()*“ е, че запълва секция от LCD екрана само с един единствен цвят.

На фиг. 5.26 е дадена и функцията „*lcd_memory_write_continue()*“, чийто поведение съвпада с това на „*lcd_memory_write()*“. Единствената разлика между двете е, че представената на фигурата функция използва командата **Write_Memory_Continue** вместо **RAMWR**. Двете команди са аналогични, но командата **Write_Memory_Continue** не връща стойностите на адресните броячи към началните (**SC** и **SP**), а продължава да записа на графичната памет от настоящия адрес. Следователно функцията „*lcd_memory_write()*“ задължително трябва да бъде употребена поне веднъж преди извикването на функцията „*lcd_memory_write_continue()*“.

```

332 void lcd_memory_write_continue(struct lcd_driver_t *driver,
333         uint16_t *data, uint32_t size)
334 {
335     _dcx_command(driver); // transmit a command
336     _csx_low(driver); // select the device
337     spi_transfer(CMD_WRITE_MEMORY_CONTINUE); // transmit the command
338     _dcx_data(driver); // transmit data
339     for (uint32_t i = 0; i < size; ++i) {
340         spi_transfer(data[i] >> 8);
341         spi_transfer(data[i] & 0xFF);
342     }
343     _csx_high(driver); // deselect the device
344 }
```

фиг. 5.26: Продължение на записването на данни в графичната памет на ILI9341

файл: „src/lcd-driver/lcd-driver.c“

Управляващият софтуер за течноокристалния еcran включва и набор от функции за рисуване на прости 2D фигури и тела. Хедър файла, в който са деклариирани функциите, е „*include/lcd-driver/graphics.h*“, като прототипите им са представени на фиг. 5.27. Функциите за рисуване използват комбинация от драйверните функции, коментирани досега в тази подточка (т. 5.4.2). Рисуващите функции имат следното предназначение:

- „*draw_pixel()*“: запълва само един пиксел от LCD екрана с определен цвят.
- „*fill_rectangle()*“: рисува запълнен с определен цвят правоъгълник върху екрана.
- „*draw_rectangle()*“: рисува незапълнен правоъгълник с определена дебелина на страните му върху екрана.
- „*draw_figure()*“: начертава 2D фигура.
- „*draw_char()*“: изрисува един ASCII символ върху LCD екрана.
- „*draw_string()*“: изрисува низ от символи върху LCD екрана.

```

20 void draw_pixel(struct lcd_driver_t *, uint16_t, uint16_t, uint16_t);
21
22 void fill_rectangle(struct lcd_driver_t *, uint16_t, uint16_t, uint16_t, uint16_t,
23                      uint16_t);
24
25 void draw_rectangle(struct lcd_driver_t *, uint16_t, uint16_t, uint16_t, uint16_t,
26                      uint16_t, uint8_t);
27
28 void draw_figure(struct lcd_driver_t *, uint16_t, uint16_t, uint16_t, uint16_t, uint16_t *);
29
30 void draw_char(struct lcd_driver_t *, uint16_t, uint16_t, unsigned char,
31                  uint16_t, uint16_t, uint8_t, fbool);
32
33 void draw_string(struct lcd_driver_t *, uint16_t, uint16_t, const char *,
34                   uint16_t, uint16_t, uint8_t, fbool);

```

фиг. 5.27: Прототипи на функциите за рисуване на прости 2D тела и фигури

файл: „*include/lcd-driver/graphics.h*“

```

11 void draw_pixel(struct lcd_driver_t *driver, uint16_t x, uint16_t y, uint16_t color)
12 {
13     // TODO: error checking for coordinates
14     lcd_column_address_set(driver, x, x);
15     lcd_page_address_set(driver, y, y);
16     lcd_memory_write(driver, &color, 1);
17 }
18
19 void fill_rectangle(struct lcd_driver_t *driver, uint16_t start_x, uint16_t start_y,
20                      uint16_t width, uint16_t height, uint16_t color)
21 {
22     lcd_column_address_set(driver, start_x, start_x + width - 1);
23     lcd_page_address_set(driver, start_y, start_y + height - 1);
24     lcd_memory_write_optimized(driver, color, (uint32_t)width * height);
25 }

```

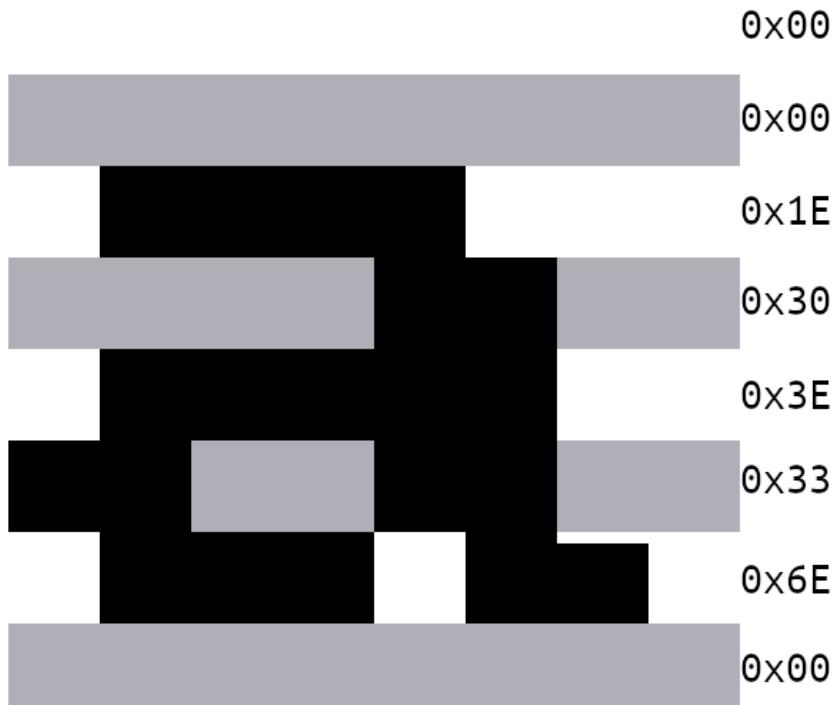
фиг. 5.28: Запълване на пиксел и запълване на правоъгълник от екрана

файл: „*src/lcd-driver/graphics.c*“

На фиг. 5.28 са представени функциите „*draw_pixel()*“ и „*fill_rectangle()*“. Имплементацията им е много сходна: определят прозореца от графичната памет, в който рисуват, след което изпращат команда за запис в паметта, запълвайки дефинирания прозорец. В случая на „*draw_pixel()*“ се запълва само един пиксел, докато функцията „*fill_rectangle()*“ запълва правоъгълна област от течнокристалния экран.

За рисуването на символи е необходимо първо да се дефинира шрифт. В настоящата дипломна работа се използва едноцветен 8×8 битмап шрифт, взет от GitHub⁽⁴¹⁾. Понеже предоставеният шрифт е част от публичния домейн, дипломантът има правото свободно да го използва за свои цели. Копие на използвания шрифт се намира във файла „*include/lcd-driver/font8x8.h*“.

Всеки един ASCII символ от използвания шрифт е съставен от 8 байта. Всеки един байт определя как ще се изрисува един ред от символа. Редът от своя страна е съставен от 8 бита, като зададените битове (в състояние „1“) се оцветяват, а останалите - не се оцветяват. На фиг. 5.29 е представен начинът на кодиране на символа „a“ в битмап шрифта.



фиг. 5.29: Символът „a“ от използвания битмап 8x8 шрифт

Шрифтът е дефиниран в друмерния масив **font** във хедър файла „*include/lcd-driver/font8x8.h*“, както е показано на фиг. 5.30. Всеки подмасив от главния масив определя точно един символ. Тъй като масивът **font** заема голямо количество памет и остава непроменен във времето, той се запазва в програмната флаш памет на микроконтролера вместо в оперативната му SRAM памет. Това се постига с макроса **PROGMEM**, сложен след декларацията на масива. Важно е да се отбележи, че при достъпване на масив в програмната памет, е необходимо да се използва някое от макросите с представка *pgm_**⁽⁴²⁾, които са дефинирани в хедър файла „*<avr/pgmspace.h>*“. На фиг. 5.31 е показано и четенето от масив в оперативната памет, и четенето от масив в програмната памет.

```

...
9  #ifndef FONT8X8_H
10 #define FONT8X8_H
11
12 #include <avr/pgmspace.h>
13
14 #define NCHARS      128
15 #define CHAR_WIDTH   8
16 #define CHAR_HEIGHT  8
17
18 #ifdef __cplusplus
19 extern "C" {
20 #endif
21
22 static const unsigned char font[NCHARS][CHAR_HEIGHT] PROGMEM = {
23     { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, // U+0000 (nul)
24     { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }    // U+007F
...
151 };
152
153 #ifdef __cplusplus
154 }
155 #endif
156
157 #endif /* FONT8X8_H */
158

```

фиг. 5.30: Запаметяване на 8×8 битмап шрифта в програмната памет на микроконтролера

файл: „*include/lcd-driver/font8x8.h*“

```

uint8_t data[10][10] = {
    ...
};

// Четене от масив в оперативната памет
uint8_t byte = data[i][j];

```

```

#include <avr/pgmspace.h>

uint8_t data[10][10] PROGMEM = {
    ...
};

// Четене от масив в програмната памет
uint8_t byte = pgm_read_byte(&data[i][j]);

```

фиг. 5.31: Четене от масив в оперативната памет (вляво) и четене от масив в програмната памет (вдясно)

На фиг. 5.32 е представена имплементацията на функцията „*draw_char()*“. Тази функция приема ASCII символ (**unsigned char**) като аргумент и го използва за да извлече своя битмап за рисуване. Итерирайки ред по ред (байт по байт от масива **font**), бит по бит, функцията запълва с определен цвят пиксели от екрана само ако провереният бит от битмата е зададен (в състояние „1“). На същата фигура е дадена и имплементацията на функцията „*draw_string()*“. Последната приема низ от символи, извиквайки функцията „*draw_char()*“ за всеки един символ, като поставя необходимото разстояние между съседни символи, за да не се припокрият.

```

54 void draw_char(struct lcd_driver_t *driver, uint16_t start_x, uint16_t start_y,
55                 unsigned char ch, uint16_t color, uint16_t bg_color, uint8_t thickness, fbool options)
56 {
57     /* fbool options:
58      * bit 0: add background color to character
59      * bits 1-7: not used
60      */
61     uint8_t row;
62
63     for (short i = 0; i < CHAR_HEIGHT; ++i) {
64         row = pgm_read_byte(&font[ch][i]);
65         for (short j = 0; j < CHAR_WIDTH; ++j) {
66             if (row & (1 << j)) {
67                 if (thickness == 1) {
68                     draw_pixel(driver, start_x + j, start_y + i, color);
69                 } else {
70                     fill_rectangle(driver, start_x + j * thickness, start_y + i * thickness,
71                                    thickness, thickness, color);
72                 }
73             } else if (options & FB00L0) {
74                 if (thickness == 1) {
75                     draw_pixel(driver, start_x + j, start_y + i, bg_color);
76                 } else {
77                     fill_rectangle(driver, start_x + j * thickness, start_y + i * thickness,
78                                    thickness, thickness, bg_color);
79                 }
80             }
81         }
82     }
83 }
84
85 void draw_string(struct lcd_driver_t *driver, uint16_t start_x, uint16_t start_y,
86                   const char *str, uint16_t color, uint16_t bg_color, uint8_t thickness, fbool options)
87 {
88     /* fbool options:
89      * bit 0: add background color to string
90      * bits 1-7: not used
91      */
92     for (short i = 0; i < strlen(str); ++i)
93         draw_char(driver, start_x + CHAR_WIDTH * i * thickness, start_y,
94                   str[i], color, bg_color, thickness, options);
95 }
```

фиг. 5.32: Рисуване на символ и на низ от символи върху LCD екрана

файл: „src/lcd-driver/graphics.c“

Разглеждайки структурата на графичната памет на драйверния чип, дадена на фиг. A.13, се забелязва, че всеки пиксел от екрана се описва с 18-битов RGB цвят. За всяка клетка са поделени 6 бита за червено, 6 бита за зелено, 6 бита за синьо и следователно може да се визуализират 262,144 уникални цвята. Въпреки това е възможно драйверната интегрална схема да се настрои в режим 16 бита/пиксел, при който всяка клетка се описва с 16-битов RGB цвят. На фиг. A.14 е показана времедиаграма от каталожните данни⁽²⁵⁾ на ILI9341, описваща начинът на обработка на 16-битов цвят в режим 16 бита/пиксел. В този режим за зеления цвят си остават заделени 6 бита, но червеният и синият цвят се описват от 5 бита. Описаният начин на интерпретиране на 16-битов цвят се нарича „565“ формат. Полученият „565“ цвят се преобразува в 18-битов цвят посредством вградена в ILI9341 таблица за справки на цветовете, като резултата се записва в адресираната клетка от графичната памет. Предимството при работа на драйверния чип в режим 16 бита/пиксел е значително ускореното предаване на данни - пренасянето на 16-битов цвят изисква две 8-битови SPI сесии, а на 18-битов цвят - три 8-битови SPI сесии. Поради ускореният пренос на данни софтуерният драйвер, разработен от дипломанта, за ILI9341 предполага, че драйверният чип е настроен в режим 16 бита/пиксел и взаимодейства само с 16-битови цветове. Недостатъкът на 16-битовия режим спрямо 18-битовия е намаленият брой уникални цветове, описани от 16-битова стойност - 65,536 цвята.

На фиг. 5.33 е представена помощната функция „color565()“, която използва три 8-битови основни цвята (червено, зелено и синьо) за образуването на 16-битов цвят във формат „565“.

```
544 uint16_t color565(uint8_t r, uint8_t g, uint8_t b)
545 {
546     return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >> 3);
547 }
```

фиг. 5.33: Образуване на 16-битов цвят във формат „565“

файл: „src/lcd-driver/lcd-driver.c“

5.4.3 Обработка на информация, получена от панела за допир

За разлика от ILI9341 драйверната интегрална схема XPT2046 за панела за допир не използва набор от команди, допълнени с параметри. Тъй като по същество интегралната схема представлява 12-битов АЦП, управляем с цифров интерфейс, е достатъчно да се окаже методът за извършване на измерване, както и измерената точка от резистивния панел за допир.

Начинът за провеждане на комуникационна сесия с XPT2046 е описан в т. 1.3.3. Според времевата диаграма, представена на фиг. 1.10, е необходимо да се изпрати контролен байт на чипа по SPI линиите, след което интегралната схема извършва желаното преобразувание и връща резултата по SPI линиите. Форматът на контролния байт е даден на фиг. 5.34 (таблица 6 от каталожните данни⁽²⁶⁾ на XPT2046). Битовете имат следното предназначение:

- **S** (бит 7): обозначава началото на комуникацията между микроконтролера и XPT2046. Този бит задължително трябва да е логическа „1“.
- **A0 ÷ A2** (битове 4 ÷ 6): избират точката от резистивния панел за допир за измерване.
- **MODE** (бит 3): определя резолюцията на следващото преобразувание. В състояние „0“ преобразуванието е 12-битово, а в „1“ - 8-битово.
- **SER/DFR** (бит 2): избира метода за осъществяване на измерването. В състояние „0“ измерването е диференциално, а в „1“ - еднотактово.
- **PD0 ÷ PD1** (битове 0 ÷ 1): определят поведението на интегралната схема в неактивно състояние, т.е. когато не се извършва преобразувание.

BIT7(MSB)	BIT 6	BIT 5	BIT 4	BIT 3	BIT2	BIT 1	BIT 0(LSB)
S	A2	A1	A0	MODE	SER/DFR	PD1	PD0

Table 6. Order of the Control Bits in the Control Byte
фиг. 5.34: Формат на контролния байт на XPT2046

На фиг. 5.35 е показана структурата **touch_driver_t**, която се използва в имплементацията на програмния код, който управлява драйверната схема XPT2046. Структурата съдържа указател към структура от тип **spi_interface_t**, както и препратки (**pin_ref_t**) към 2-те допълнителни сигнални линии, чийто предназначение е:

- **TCS** (**tcs** в кода): обозначава драйверния чип XPT2046 като адресанта на SPI комуникацията. Активен при ниско логическо ниво.
- **PENIRQ** (**tirq** в кода): при допир на панела драйверният чип генерира прекъсване което е изпратено на този извод. Прекъсването е обозначено с ниско логическо ниво.

```

...
25 // deselect the Touch Panel driver (output a HIGH level to Chip Select)
26 #define _tcs_high(touch_driver) { \
27     *touch_driver->tcs.port |= (1 << touch_driver->tcs.pin); \
28 }
29
30 // select the Touch Panel driver (output a LOW level to Chip Select)
31 #define _tcs_low(touch_driver) { \
32     *touch_driver->tcs.port &= ~(1 << touch_driver->tcs.pin); \
33 }
34
35 // get the state of the Pen Interrupt output
36 #define _tirq_state(touch_driver) (*touch_driver->tirq.port & _BV(touch_driver->tirq.pin))
...
42 struct touch_driver_t {
43     struct spi_interface_t *spi; // SPI interface
44     struct pin_ref_t tcs; // Chip Select
45     struct pin_ref_t tirq; // Pen Interrupt
46 };
47
48 void touch_driver_init(struct touch_driver_t *, fbool);

```

фиг. 5.35: Дефиниция на структурата **touch_driver_t**

файл: „include/touch-panel-driver/touch-panel-driver.h“

За управление на **TCS** сигнала е дефинирана двойката макроси „_tcs_high()“ и „_tcs_low()“, които променят логическото ниво на извода. За прочитане на състоянието на **PENIRQ** сигнала се използва макроса „_tirq_state()“. Споменатите макроси значително подобряват качеството на имплементационния код на управляващия софтуер за XPT2046. Въпреки това тези макроси не трябва да се употребяват извън имплементацията на драйвера за панела за допир.

На фиг. 5.36 е представена функцията, която инициализира променлива от тип **touch_driver_t**. Функцията позволява инициализацията на SPI порта при продаване на ненулева стойност на параметъра **options**. Освен това микроконтролерът поставя драйверния чип в неактивно състояние (**TCS** = „1“).

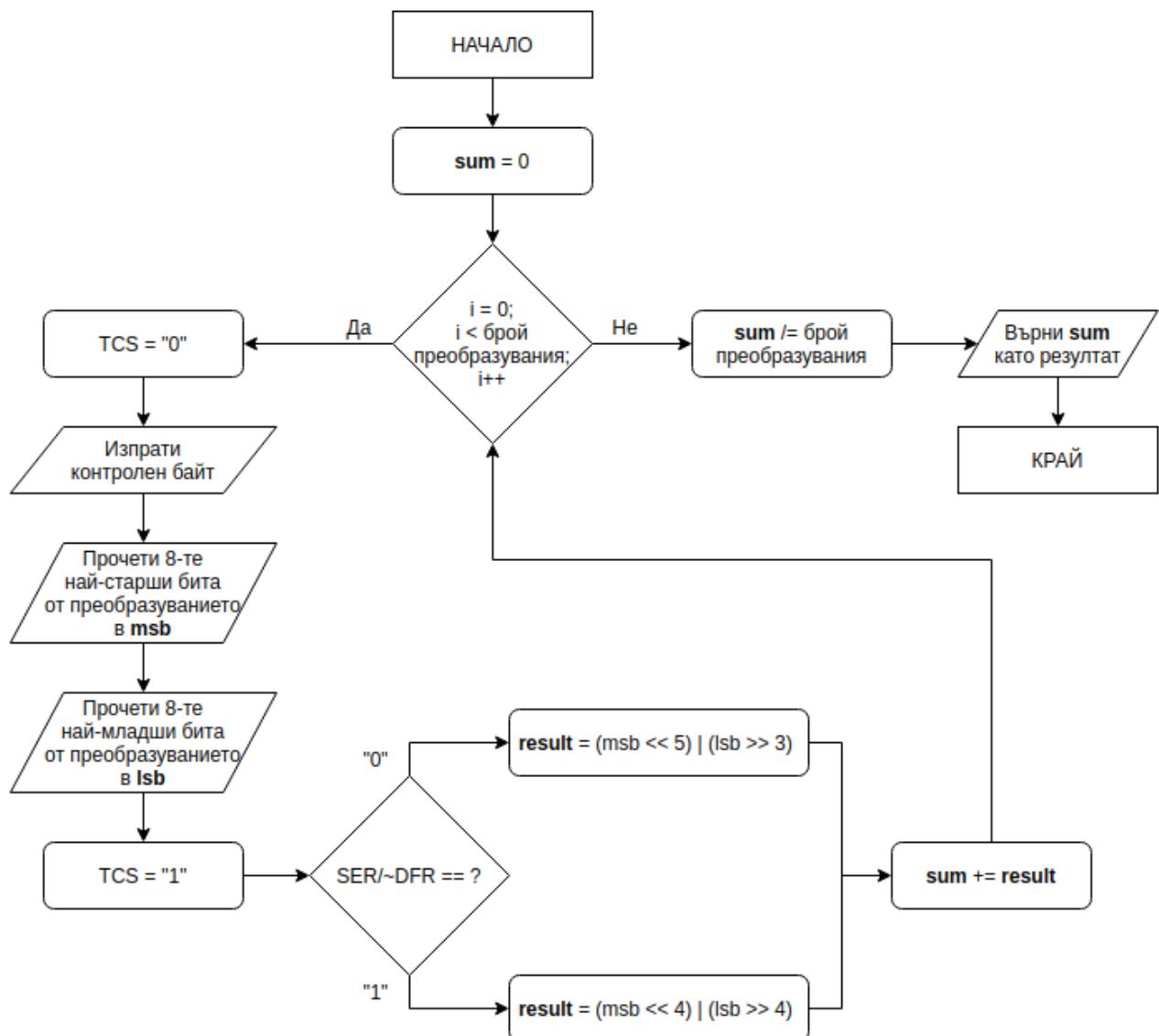
```

16 void touch_driver_init(struct touch_driver_t *driver, fbool options)
17 {
18     /* fbool options:
19      * bit 0: initialize Master SPI
20      * bits 1-7: not used
21      */
22     // optionally initialize the SPI interface
23     if (options & FB00L0)
24         mspi_init(driver->spi);
25
26     _tcs_high(driver);
27 }

```

фиг. 5.36: Инициализация на структурата **touch_driver_t**

файл: „src/touch-panel-driver/touch-panel-driver.c“



фиг. 5.37: Блокова схема на алгоритъма за прочитане на координат от панела за допир

Алгоритъмът за прочитане на координат при допир от панела се описва по блоковата схема, показана на фиг. 5.37. За извършване на еднократно преобразувание е необходимо първо да се адресира XPT2046 (**TCS = „0“**), след което да се изпрати правилно форматиран контролен байт. Драйверният чип извършва преобразуванието и изпраща резултата обратно на микроконтролера с две 8-битови SPI сесии. Микроконтролерът спира комуникацията с интегралната схема (**TCS = „1“**) и сглобява резултата от преобразуванието, използвайки двета приети байта по SPI. Ако преобразуванието е диференциално (**SER/D⁺FR = „0“**), драйверният чип XPT2046 се нуждае от един допълнителен такт, за да извърши преобразуванието⁽²⁶⁾. Затова най-старият приет бит се пренебрегва при сглобяване на резултата. В противен случай (**SER/D⁺FR = „1“**) резултатът се сглобява от първите 12 приети бита. Процесът по прочитане на координат, т.е. сдобиване с преобразувание, от XPT2046 се повтаря

няколкократно, след което се извършва средно аритметично преобразуване на всички получени преобразования. Извършвайки няколко преобразования, резултатът от алгоритъма е по-прецизен и по-надежден спрямо резултат, получен от единствено преобразуване.

Описаният алгоритъм от фиг. 5.37 е имплементиран от двете статични функции, представени на фиг. 5.38. Функцията „_touch_read_ADC()“ реализира успешна комуникационна сесия между микроконтролера и драйверния чип и обработва получените данни. Функцията „_touch_multiread()“ извършва средно аритметично преобразуване на няколко резултата, получени чрез извикване на функцията „_touch_read_ADC()“.

```

91 static uint16_t _touch_read_ADC(struct touch_driver_t *driver, uint8_t cmd)
92 {
93     uint8_t lsb, msb;
94
95     _tcs_low(driver);           // select the device
96     spi_transfer(cmd);        // transfer the control byte
97     msb = spi_transfer(0x00);  // read the 8 most significant bits
98     lsb = spi_transfer(0x00);  // read the 8 least significant bits
99     _tcs_high(driver);        // deselect the device
100    if (cmd & TP_SERDFR) // single-ended conversion (12 clk cycles)
101        return (msb << 4) | (lsb >> 4);
102    else // ratiometric conversion (13 clk cycles)
103        return (msb << 5) | (lsb >> 3);
104 }
105
106 static uint16_t _touch_multiread(struct touch_driver_t *driver, uint8_t cmd)
107 {
108     uint32_t sum = 0;
109
110     for (short i = 0; i < SAMPLES; ++i)
111         sum += _touch_read_ADC(driver, cmd);
112     return sum / SAMPLES;
113 }
```

фиг. 5.38: Реализация на алгоритъма за прочитане на координат от панела за допир

файл: „src/touch-panel-driver/touch-panel-driver.c“

Важно е да се отбележи, че получените данни от драйверния чип XPT2046 представляват преобразования на АЦП, т.е. цифровизирано ниво на напрежение. За комбиниране на панела за допир с течнокристалния екран е необходимо получените данни да се преобразуват към координати (т.е. единствен пиксел) от экрана. На фиг. 5.39 са представени две аналогични функции, които преобразуват резултата от измерванията на драйверния чип в координати на экрана. Тази операция се извършва с помощта на калибрационните параметри, зададени в макросите **XFAC**, **XOFFSET**, **YFAC** и **YOFFSET**. Както и методът за преобразуване на данни от XPT2046 към координати на экрана, така и самите калибрационни параметри са взети от библиотеката „*LCDWIKI_TOUCH*“ на LCDWIKI. На фиг. A.18 е изведен откъс от програмния код на библиотеката, който изпълнява преобразуванието.

```

29 uint16_t touch_read_x(struct touch_driver_t *driver)
30 {
31     return ((long)XFAC * _touch_multiread(driver, TP_CMD_RDX)) / 10000 + XOFFSET;
32 }
33
34 uint16_t touch_read_y(struct touch_driver_t *driver)
35 {
36     return ((long)YFAC * _touch_multiread(driver, TP_CMD_RDY)) / 10000 + YOFFSET;
37 }

```

фиг. 5.39: Преобразуване на данни от XPT2046 към координати на LCD екрана

файл: „src/touch-panel-driver/touch-panel-driver.c“

На фиг. 5.40 е показана функцията „*touch_scan()*“, чието предназначение е да засече допир и да прочете координатите на допирната точка. Първо се проверява състоянието на **PENIRQ** сигнала, който сигнализира допир с ниско логическо ниво. Следователно при прочитане на състояние „0“ микроконтролерът стартира комуникация с драйверния чип, прочитайки координатите на допирната точка, и функцията връща ненулева стойност. Ако състоянието на **PENIRQ** сигнала е „1“, функцията връща нулева стойност.

```

39 short touch_scan(struct touch_driver_t *driver, uint16_t *x, uint16_t *y)
40 {
41     if (!_tirq_state(driver)) { // touch panel is pressed
42         *x = touch_read_x(driver);
43         *y = touch_read_y(driver);
44         return 1;
45     }
46     return 0;
47 }

```

фиг. 5.40: Засичане на допир и прочитане на координатите на допирната точка

файл: „src/touch-panel-driver/touch-panel-driver.c“

Описаният досега управляващ код осъществява успешна комуникация с драйверната интегрална схема XPT2046 за панела за допир и прочита координатите на допирната точка, но не взема под внимание начинът, по който се манипулира графичната памет на интегралната схема ILI9341, управляваща екрана. Следователно за да може правилно да се употреби информацията, получена от XPT2046, е необходимо управляващият софтуер да вземе под внимание стойността на регистъра **MADCTL** на ILI9341 при използване на резултатите от функцията „*touch_scan()*“ за взаимодействие с течноокристалния екран. Дипломантът опитно е установил, че течноокристалният екран и панела за допир са ориентирани противоположно един спрямо друг, когато стойността на регистъра **MADCTL** е 0_{16} (стойността по подразбиране), както е показано на фиг. 5.41. Графичната памет на ILI9341

се манипулира от горе на долу, от ляво на дясно, докато чипът XPT2046 отчита координатите на панела от долу на горе, от дясно на ляво спрямо ILI9341.



фиг. 5.41: Взаимоотношение между XPT2046 и ILI9341 при отчитането на координати върху экрана (**MADCTL** = 0_{16})

За да може да се синхронизират получените данни от XPT2046 с манипулацията на графичната памет на ILI9341, се дефинира своеобразно ориентация на течнокристалния екран спрямо стойността по подразбиране (0_{16}) на регистъра **MADCTL**. Приема се, че ако графичната памет се манипулира от горе на долу (**MADCTL.MY** = "0"), към ориентацията се добавя ключовата дума „NORTH“. В противен случай (**MADCTL.MY** = "1") се добавя „SOUTH“. Също така ако графичната памет се манипулира от ляво на дясно (**MADCTL.MX** = "0"), се добавя ключовата дума „WEST“, а в противен случай (**MADCTL.MX** = "1") - „EAST“. Двете ключови думи се добавят една към друга, като при (**MADCTL.MV** = "0") първо се добавя думата „NORTH“/„SOUTH“, а при (**MADCTL.MV** = "1") - „WEST“/„EAST“. Съчетавайки двете ключови думи, се дефинира ориентация, която идентифицира началната точка (ред 0-ев, колона 0-ева), от която се манипулира графичната памет. Идеята за дефиниране на ориентация на экрана е илюстрирана на фиг. 5.42 и на фиг. 5.43. Оцветените точки означават началната точка (ред 0-ев, колона 0-ева), от която започва запис в графичната памет, на съответната ориентация (оцветена със същия цвят).



фиг. 5.42: Възможни ориентации на екрана при **MADCTL.MV** = „0“

фиг. 5.43: Възможни ориентации на екрана при **MADCTL.MV** = „1“

В програмния код ориентацията на екрана се обозначава с енумерацията **rotation**, показана на фиг. 5.44. Важно е да се отбележи, че числовата стойност на ориентациите при **MADCTL.MV** = „1“ е отместена с +4 спрямо еквивалентните им ориентации при **MADCTL.MV** = „0“. Този факт, както и настоящата стойност на регистъра MADCTL, се използват от функцията „*lcd_get_rotation()*“, която е показана на фиг. 5.45, за да се определи ориентацията на екрана.

```

53 enum rotation {
54     NORTHEAST = 0,
55     SOUTHWEST = 1,
56     NORTHWEST = 2,
57     SOUTHEAST = 3,
58     EASTNORTH = 4,
59     WESTSOUTH = 5,
60     WESTNORTH = 6,
61     EASTSOUTH = 7,
62 };

```

фиг. 5.44: Числови стойности на всяка възможна ориентация на екрана

файл: „src/lcd-driver/lcd-driver.c“

```

549 enum rotation lcd_get_rotation(struct lcd_driver_t *driver)
550 {
551     enum rotation result;
552
553     if (driver->madctl & MADCTL_MY) { // bottom to top row order
554         if (driver->madctl & MADCTL_MX) // right to left column order
555             result = SOUTHEAST;
556         else // left to right column order
557             result = SOUTHWEST;
558     } else { // top to bottom row order
559         if (driver->madctl & MADCTL_MX) // right to left column order
560             result = NORTHEAST;
561         else // left to right column order
562             result = NORTHWEST;
563     }
564     if (driver->madctl & MADCTL_MV) // row/column order is exchanged
565         result += 4; // exchange x/y directions
566
567     return result;
568 }
```

фиг. 5.45: Определяне на ориентацията на екрана

файл: „src/lcd-driver/lcd-driver.c“

Знаеики своеобразно определената ориентация на екрана, е възможно координатите, получени от драйверната схема за панела за допир, да се асоциират с определена клетка от графичната памет на ILI9341 (и съответно към определен пиксел от течнокристалния екран) независимо от начина ѝ на манипулация, т.е. независимо от стойността на регистъра **MADCTL**. Това асоцииране се извършва от функцията „*touch_get_screen_coordinates()*“, показана на фиг. 5.46. Функцията извършва нужните преобразования на координатите и записва резултатите в променливите, посочени от указателите **x** и **y**, подадени като нейни аргументи.

```

49 void touch_get_screen_coordinates(struct lcd_driver_t *lcd_driver,
50         uint16_t *x, uint16_t *y)
51 {
52     uint16_t temp;
53     enum rotation screen_rotation = lcd_get_rotation(lcd_driver);
54
55     switch (screen_rotation) {
56     case NORTHEAST:
57         *y = lcd_driver->res_y - *y;
58         break;
59     case SOUTHWEST:
60         *x = lcd_driver->res_x - *x;
61         break;
62     case NORTHWEST:
63         *x = lcd_driver->res_x - *x;
64         *y = lcd_driver->res_y - *y;
65         break;
66     case SOUTHEAST:
67         break;
68     case EASTNORTH:
69         temp = *x;
70         *x = lcd_driver->res_x - *y;
71         *y = temp;
72         break;
73     case WESTSOUTH:
74         temp = *x;
75         *x = *y;
76         *y = lcd_driver->res_y - temp;
77         break;
78     case WESTNORTH:
79         temp = *x;
80         *x = lcd_driver->res_x - *y;
81         *y = lcd_driver->res_y - temp;
82         break;
83     case EASTSOUTH:
84         temp = *x;
85         *x = *y;
86         *y = temp;
87         break;
88     }
89 }

```

фиг. 5.46: Асоцииране на координати, получени от XPT2046, към определи клетки от графичната паме на ILI9341

файл: "src/touch-panel-driver/touch-panel-driver.c"

Съчетавайки описаните в тази точка функции, на фиг. 5.47 е представен примерен код, който следи състоянието на панела за допир. Ако последният е натиснат, на екрана се оцветява пикселът, който съвпада с допирната точка.

```

int main(void)
{
    struct lcd_driver_t lcd;
    struct touch_driver_t touch;
    // Инициализация на микроконтролера, ILI9341 и XPT2046

    uint16_t x, y;
    uint16_t color;
    while (1) {
        if (touch_scan(&touch, &x, &y)) {
            touch_get_screen_coordinates(&lcd, &x, &y);
            draw_pixel(&lcd, x, y, color);
        }
    }
}

```

фиг. 5.47: Примерен код, който оцветява докоснатите от потребителя пиксели на екрана

5.5 Алгоритъм за управление на устройството

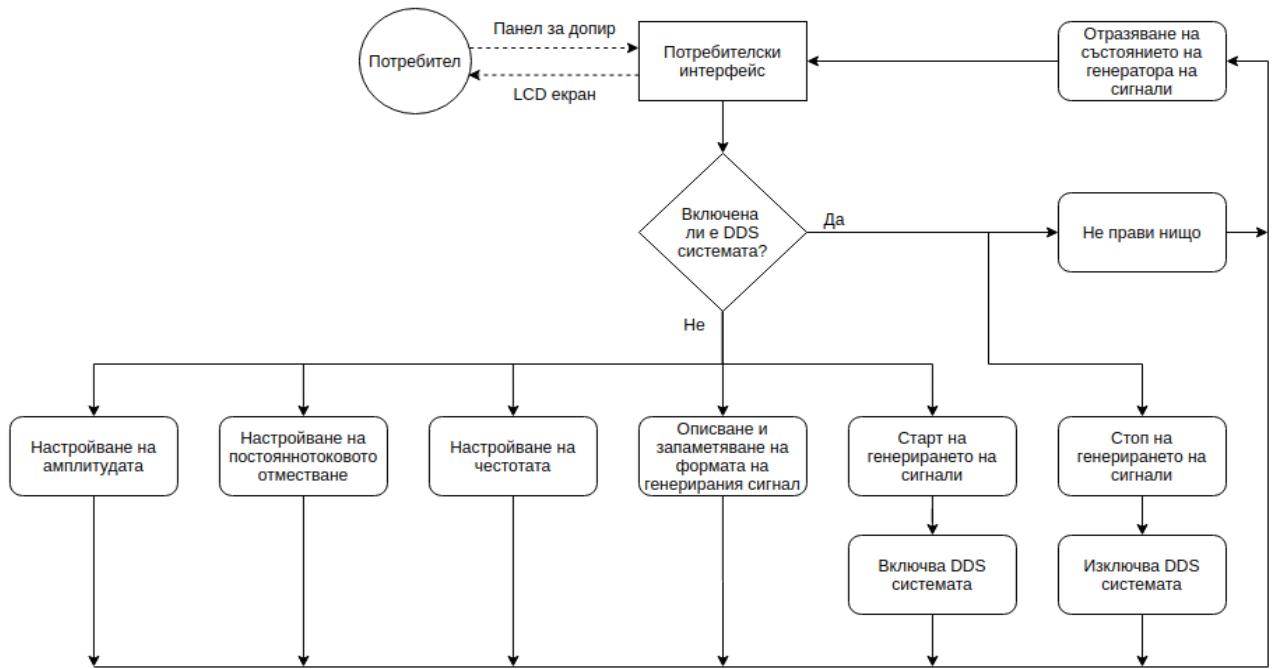
За да се постигне успешно и параметризирано генериране на сигнал с произволна форма, управляващият софтуер на микроконтролера трябва само да конфигурира интегралните схеми, съставящи цифровия генератор на сигнали, без да има пряка намеса в процеса на генериране.

5.5.1 Блокова диаграма на управляващия устройството алгоритъм

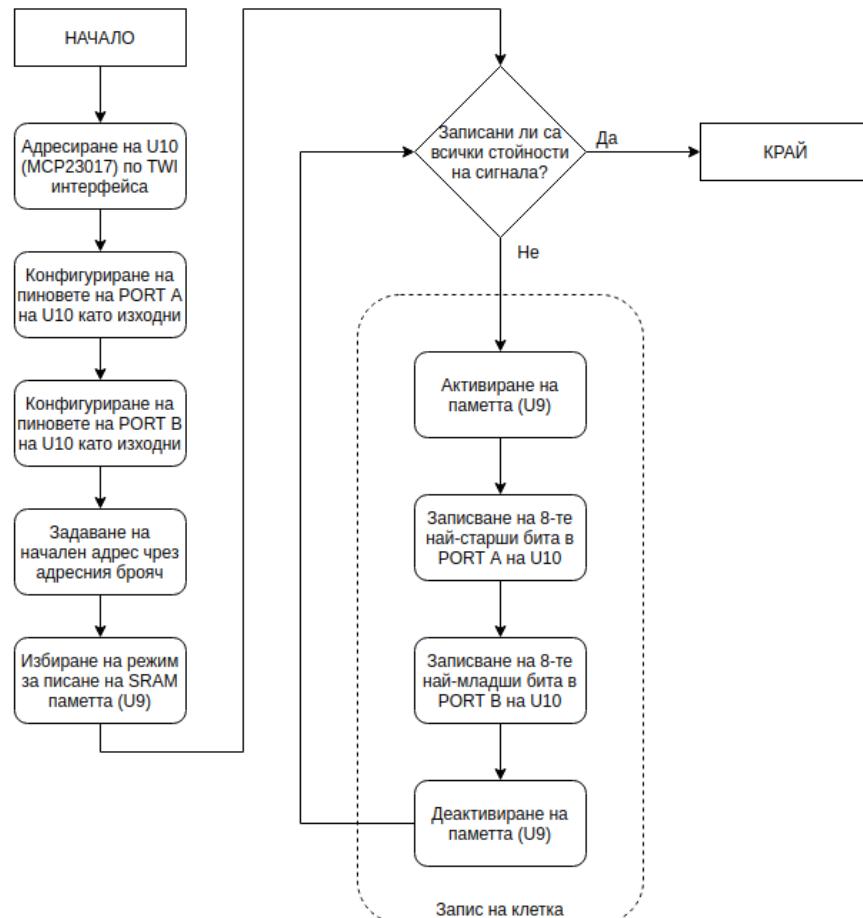
Блоковата диаграма на управляващия алгоритъм за цифровия генератор на сигнали е представена на фиг. 5.48. Потребителят взаимодейства с устройството посредством потребителския интерфейс, реализиран на течнокристалния екран. При приемана на потребителски вход (докосване на панела за допир) микроконтролерът проверява дали е включена DDS системата. Ако е включена, потребителят има единствено избора да спре генерирането на сигнали. Алгоритъмът не позволява параметризиране на генеририания сигнал при работеща DDS система. Ако последната не е включена, потребителят има възможността и да промени формата на сигнала, и да параметризира основните му характеристики - амплитуда, постояннотоково отместване, честота, и да включи DDS системата. Каквото и действие да предприеме потребителят, чрез течнокристалния екран потребителският интерфейс отразява състоянието на генератора на сигнали без значение дали е включена или изключена DDS системата.

5.5.2 Генериране на електрически сигнал с произволна форма

Първоначално е необходимо формата на желания електрически сигнал да се запамети в SRAM паметта (**U9**) на DDS системата. На фиг. 5.49 е показана блоковата схема на алгоритъма за записване на желаната форма на генеририания сигнал в паметта.



фиг. 5.48: Блокова диаграмма на управляващия софтуер за цифровия генератор на сигнали



фиг. 5.49: Блокова схема на алгоритъма за записване на паметта на DDS системата

От фиг. 5.49 стана ясно, че стойностите, предназначени за запис в клетките на паметта, се предават през преходника последователна-паралелна комуникация MCP23017 (**U10**). Програмирането на MCP23017 се състои в конфигурирането на вътрешните му регистри (показани на фиг. A.4) посредством I²C интерфейс. На фиг. A.15 е представен форматът за записване на стойности в регистрите на MCP23017. В контролния байт се подава I²C адреса на **U10** и сигнал за запис (**R/W** = „0“). В първия кадър за данни се посочва адресът на конфигурирания регистър, след което в следващия I²C кадър се подава желаната стойност за записване в регистъра. След извършване на операцията вътрешният адресен брояч на преходника започва да сочи към следващия поред регистър на MCP23017. Поредността и адресите на регистрите зависи от бита **BANK** на регистъра **IOCON** (виж фиг. A.4). След настройване на адресирания регистър интегралната схема (ATmega2561), управляваща преходника, има избора да приключи комуникацията или да продължи да записва стойности на следващите поред регистри.

Преди да може да се предадат данни през **U10** е необходимо пиновете на двата му порта - **PORT A** и **PORT B**, да се конфигурират като изходни пинове. Посоките (входни или изходни) на пиновете се определят от стойностите, записани в регистрите **IODIRA** и **IODIRB**, чийто формат е показан на фиг. 5.50, взета от каталожните данни⁽¹⁴⁾ на MCP23017. Всеки бит от регистрите определят посоката на съответния пин на преходника. Ако битът е зададен (в състояние „1“) съответният пин е високоомен вход. В противен случай (в състояние „0“) пинът се държи като нискоомен изход. На практика задаването на посоките на пиновете се извършва от функцията „*ioex_set_iodir()*“, показана на фиг. 5.51. Чрез параметъра **options** се задава желаният за конфигурация регистър и се оказва настоящето състояние на бита **IOCON.BANK**. Макросите с представка „*MCP23017_**“ са дефинирани във файла „*include/DFG-firmware/chips/MCP23017.h*“ с цел подобряване качеството на кода.

REGISTER 3-1: IODIR: I/O DIRECTION REGISTER (ADDR 0x00)

| R/W-1 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| IO7 | IO6 | IO5 | IO4 | IO3 | IO2 | IO1 | IO0 |
| bit 7 | | | | | | | bit 0 |

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared

bit 7-0 **IO<7:0>**: Controls the direction of data I/O <7:0>
 1 = Pin is configured as an input.
 0 = Pin is configured as an output.

фиг. 5.50: Формат на регистрите **IODIRA** и **IODIRB** на MCP23017

```

18 uint8_t ioex_set_iendir(uint8_t address, uint8_t dir, fbool options)
19 {
20     /* fbool options:
21      * bit 0: IOCON.BANK bit
22      * bit 1: port - '0' for port A
23      *           - '1' for port B
24      * bits 2-7: not used
25      */
26     uint8_t twi_data[2];
27
28     twi_data[1] = dir;
29     if (options & FB00L0) // IOCON.BANK = 1
30         twi_data[0] = (options & FB00L1) ? MCP23017_B1_IODIRB : MCP23017_B1_IODIRA;
31     else // IOCON.BANK = 0
32         twi_data[0] = (options & FB00L1) ? MCP23017_B0_IODIRB : MCP23017_B0_IODIRA;
33     return twi_write(address, twi_data, 2);
34 }

```

фиг. 5.51: Задаване на посоката на пиновете на регистъра **IODIRA** или **IODIRB**

файл: "src/DFG-firmware/IOexpander_driver.c"

След като се конфигурират пиновете на **PORT A** и **PORT B** като изходни, е възможно да се настрои логическото им ниво посредством регистрите **GPIOA** и **GPIOB**. Форматът им е описан на фиг. 5.52, взета от каталожните данни⁽¹⁴⁾ на MCP23017. Логическото изходно ниво на всеки пин съвпада със състоянието на съответния му бит. Функцията „*ioex_write_gpio()*“, показана на фиг. 5.53, се изполва за промяна на логическите изходни нива на пиновете. Тя е аналогична на функцията „*ioex_set_iendir()*“ (фиг. 5.51).

REGISTER 3-10: GPIO: GENERAL PURPOSE I/O PORT REGISTER (ADDR 0x09)

| R/W-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| GP7 | GP6 | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 |
| bit 7 | | | | | | | bit 0 |

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

bit 7-0 **GP<7:0>**: Reflects the logic level on the pins <7:0>
 1 = Logic-high
 0 = Logic-low

фиг. 5.52: Формат на регистрите **GPIOA** и **GPIOB** на MCP23017

```

36 uint8_t ioex_write_gpio(uint8_t address, uint8_t data, fbool options)
37 {
38     /* fbool options:
39      * bit 0: IOCON.BANK bit
40      * bit 1: port - '0' for port A
41      *           - '1' for port B
42      * bits 2-7: not used
43      */
44     uint8_t twi_data[2];
45
46     twi_data[1] = data;
47     if (options & FB00L0) // IOCON.BANK = 1
48         twi_data[0] = (options & FB00L1) ? MCP23017_B1_GPIOB : MCP23017_B1_GPIOA;
49     else // IOCON.BANK = 0
50         twi_data[0] = (options & FB00L1) ? MCP23017_B0_GPIOB : MCP23017_B0_GPIOA;
51     return twi_write(address, twi_data, 2);
52 }

```

фиг. 5.53: Промяна на изходните логически нива на пиновете на PORT A и PORT B

файл: "src/DFG-firmware/IOexpander_driver.c"

Но за да се запише стойност в клетка от SRAM паметта, е необходимо да се укаже и нейният адрес, което се постига чрез настройване на 16-битовия адресен брояч на DDS системата. Задаването на адрес от паметта се изпълнява от функцията „*set_address_counter()*“, дадена на фиг. 5.54. Първоначално се нулира стойността на адресния брояч, след което се инкрементира докато не достигне желания адрес, указан от параметъра **address**.

```

26 void set_address_counter(uint16_t address, struct pin_ref_t cp,
27     struct pin_ref_t cmrl, struct pin_ref_t cmr2)
28 {
29     setpinref(cp);
30     reset_address_counter(cmrl, cmr2);
31     for (uint16_t i = 0; i < address; ++i)
32         increment_address_counter(cp);
33 }

```

фиг. 5.54: Настройване на 16-битовия адресен брояч на DDS системата

файл: "src/DFG-firmware/DDS_firmware.c"

На фиг. 5.55 са дадени и две допълнителни функции - „*power_up()*“ и „*power_down()*“, които съответно включват и изключват интегралните схеми, закачени към **PWRDWN** линията. След включване („*power_up()*“) е необходимо да се изчака известно време за активирането на всички схеми.

```

10 void power_up(struct pin_ref_t pwrdown)
11 {
12     resetpinref(pwrdown);
13     _delay_us(POWER_UP_TIME);
14 }
15
16 void power_down(struct pin_ref_t pwrdown)
17 {
18     setpinref(pwrdown);
19 }

```

фиг. 5.55: Включване и изключване на DDS системата

файл: "src/DFG-firmware/DDS_firmware.c"

Функцията „*sram_write()*“, показана на фиг. 5.56, използва функциите, които контролират DDS системата, за да имплементира алгоритъма от фиг. 5.49. Първоначално се конфигурират всички пинове на **PORT A** и **PORT B** на MCP23017 като изходни и паметта се поставя в режим на запис (69-ти ред). Паметта се активира, записва се желаната 16-битова стойност в първата клетка, деактивира се чипът и накрая се инкрементира стойността на адресния брояч, като по този начин се адресира следващата клетка. Описаният процес се повтаря, докато последователно се запишат всички желани стойности в SRAM паметта. Важно е да се отбележи, че функцията **НЕ** нулира или задава първоначалната стойност на адресния брояч. Следователно е необходимо броячът да посочи първия адрес предварително.

```

61 uint8_t sram_write(uint16_t *data, size_t size, struct pin_ref_t rw,
62                     struct pin_ref_t pwrdown, struct pin_ref_t cp)
63 {
64     uint8_t rc;
65
66     // set all pins on both PORT A and PORT B as outputs
67     if ((rc = ioex_set_iodir(U10_TWI_ADDRESS, 0x00, 0)) != 0) return rc;
68     if ((rc = ioex_set_iodir(U10_TWI_ADDRESS, 0x00, FB00L1)) != 0) return rc;
69     resetpinref(rw);                                         // write to SRAM
70     for (size_t i = 0; i < size; ++i) {
71         power_up(pwrdown);                                     // activate the chip
72         rc = ioex_write_gpio(U10_TWI_ADDRESS, data[i] & 0xFF, FB00L1); // write LSBs to GPIOB
73         if (rc != 0) return rc;
74         rc = ioex_write_gpio(U10_TWI_ADDRESS, data[i] >> 8, 0);    // write MSBs to GPIOA
75         if (rc != 0) return rc;
76         power_down(pwrdown);                                     // deselect the chip
77         increment_address_counter(cp);                           // move to next memory cell
78     }
79     return 0;
80 }

```

фиг. 5.56: Записване на поредица от стойности в паметта на DDS системата

файл: "src/DFG-firmware/DDS_firmware.c"

За да работи правилно DDS системата, е нужно стойностите от SRAM паметта да бъдат прочетени и достъпени от ЦАП-а на системата. Функцията „*load_into_dac()*“, която е показана на фиг. 5.57, изпълнява нужните приготовления и установява директна връзка

между ЦАП-а и паметта. Всички пинове на MCP23017 задължително тряба да се настроят като входни. В противен случай те биха оказали влияние на предаваните от паметта данни. **U9** се поставя в режим на четене (ред 89-ти), след което се включва системата.

```

82 uint8_t load_into_dac(struct pin_ref_t *rw, struct pin_ref_t *pwrdown)
83 {
84     uint8_t rc;
85
86     // set all pins on both PORT A and PORT B as inputs
87     if ((rc = ioex_set_iodir(U10_TWI_ADDRESS, 0xFF, 0)) != 0) return rc;
88     if ((rc = ioex_set_iodir(U10_TWI_ADDRESS, 0xFF, FB00L1)) != 0) return rc;
89     setpinref(rw);           // read from SRAM
90     power_up(pwrdown);      // activate the chip
91     return 0;
92 }
```

фиг. 5.57: Зареждане на стойностите от SRAM паметта в ЦАП-а на DDS системата

файл: „src/DFG-firmware/DDS_firmware.c“

Последната интегрална схема, която тряба да се настрои за стартиране на процеса на генериране на сигнал, е програмируемият осцилатор DS1085, означен на фиг. 3.33 с **U8**. Програмирането на тази интеграна схема също се извършва през I²C интерфейс. DS1085 разполага с няколко регистра, някои с дължина един байт, някои с дължина два байта, и съответна команда за достъпването им. Различните видове комуникационни сесии, нужни за програмирането на осцилатора, са показани на фиг. A.16.

Важно е да се отбележи, че паметта, от която са съставени регистрите, е енергонезависима, поради което се установяват следните наблюдения:

- 1) Конфигурацията на DS1085 не се губи при откачане на захранването му.
- 2) По подразбиране след успешна обработка на I²C команда интегралната схема автоматично записва новата конфигурация в паметта. Този процес отнема известно минимално време за изпълняването му, като в каталожните данни⁽¹³⁾ е посочена стойността 10ms.

Достатъчно е да се настрои единствено 16-битовия регистър **MUX WORD**⁽¹³⁾, за да се нагоди DS1085 към принципната схема от фиг. 3.33. Форматът на регистъра е показан на фиг. 5.58, взета от каталожните данни⁽¹³⁾ на DS1085. Най-старшият бит задължително трява да бъде „0“, а шестте най-младши бита се пренебрегват и нямат функция. Битът **PDN1** определя дали входът **CTRL1** на DS1085 се използва за включване/изключване на изхода **OUT1** или за включване/изключване на цялото устройство (виж фиг. A.2). Тъй като в принципната схема е предназначено входът **CTRL1** да се използва за контролиране само на изхода **OUT1**, този бит приема стойността „0“ в управляващия софтуер. Различните комбинации от състояния на битовете **PDN0**, **SEL0** и **EN0** определят различни функции за входа **CTRL0**, описани в

таблица 2 (виж фиг. А.3) от каталожните данни⁽¹³⁾ на DS1085. Тъй като предназначението на входа **CTRL0** е да се използва за включване/изключване на цялото устройство, битът **PDN0** заема стойност „1“, състоянието на бита **SEL0** зависи от това дали се използва разделителя на честота (от англ.: „prescaler“) на изхода **OUT0**, а състоянието на бита **EN0** е без значение.

MUX WORD (Address 02h)

The MUX word controls several functions. Its bits are organized as follows:

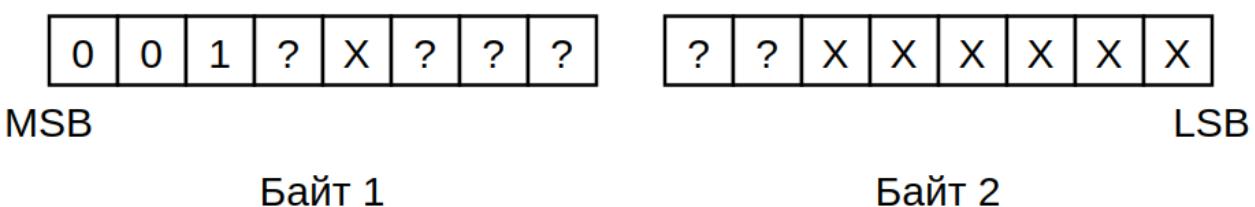
The MAC word controls several functions. Its bits are organized as follows:															
MSB						LSB			MSB			LSB			
NAME	*	PDN1	PDN0	SEL0	EN0	0M1	0M0	1M1	1M0	DIV1	-	-	-	-	-
Default Setting	0	0	0	1	1	0	0	0	0	0	X	X	X	X	X

*This bit must be set to zero.

$X \equiv$ Don't care

фиг. 5.58: Формат на регистъра **MUX WORD** на DS1085

Следователно стойността, подаден на регистъра **MUX WORD**, трябва да се изгради по следния образец (X - без значение, ? - определ от управляващата програма):



фиг. 5.59: Образец за стойност на **MUX WORD** регистъра

Конфигурацията на регистъра **MUX WORD** се извършва от функцията „*osc_conf_mux_word()*“, показана на фиг. 5.60. Тя очаква стойността на параметъра **mux_word** да е по образец от фиг. 5.59. Командата за програмиране на регистъра, както и новата му стойност се изпращат чрез TWI интерфейса на микроконтролера, след което последният задължително изчаква необходимото време за запис на новата стойност в енергонезависимата памет на DS1085.

```
20 uint8_t osc_conf_mux_word(uint8_t address, uint16_t mux_word)
21 {
22     // mux_word is assumed to be in format bbbbbbbb bbxxxxxx
23     uint8_t access_mux[3] = { DS1085_ACCESS_MUX, (mux_word >> 8), (mux_word & 0xFF) };
24     uint8_t rc;
25
26     if ((rc = twi_write(address, access_mux, 3)) != 0) return rc;
27     _delay_ms(10);
28     return 0;
29 }
```

фиг. 5.60: Програмиране на регистъра **MUX WORD** на DS1085

файл: ..src/DEG-firmware/programmable oscillator driver.c“

На фиг. 5.61 са показани функциите, с които се активира и деактивира програмируемия осцилатор, манипулирайки състоянието на проводящата линия **OSCOE**.

```
10 void osc_output_enable(struct pin_ref_t oscoe)
11 {
12     resetpinref(oscoe);
13 }
14
15 void osc_output_disable(struct pin_ref_t oscoe)
16 {
17     setpinref(oscoe);
18 }
```

фиг. 5.61: Активиране и деактивиране на програмируемия осцилатор

файл: „src/DFG-firmware/programmable_oscillator_driver.c“

```
#define MUX_WORD
#define DATA_SIZE

int main(void)
{
    // Конфигурира пин ~CP като изходен и му задава високо логическо ниво
    DDRB |= _BV(PB7);
    PORTB |= _BV(PB7);

    // Настройване на програмируемия осцилатор DS1085
    struct pin_ref_t oscoe = { &PORTF, PF2 };
    osc_conf_mux_word(U8_TWI_ADDRESS, MUX_WORD);
    osc_output_disable(oscoe);

    // Използваните пинове за управление на DDS системата
    struct pin_ref_t pwrdsn = { &PORTF, PFO };
    struct pin_ref_t cmr1 = { &PORTB, PB4 };
    struct pin_ref_t cmr2 = { &PORTB, PB5 };
    struct pin_ref_t cp = { &PORTB, PB7 };
    struct pin_ref_t rw = { &PORTB, PB6 };

    // Данните за записване в SRAM паметта на DDS системата
    // Бележка: размерът на масива с данни (DATA_SIZE) задължително
    // трябва да бъде степен на 2
    uint16_t data[DATA_SIZE] = {
        ...
    };
    // Неколократно записване на данните в паметта
    for (uint16_t i = 0; i < (uint16_t)pow(2, 16) / DATA_SIZE; ++i) {
        set_address_counter(i * DATA_SIZE, cp, cmr1, cmr2);
        sram_write(data, DATA_SIZE, rw, pwrdsn, cp);
    }
    // Зареждане на данните от паметта в ЦАП-а
    load_into_dac(rw, pwrdsn);

    // Нулиране на брояча
    reset_address_counter(cmr1, cmr2);
    // Настройване на пин ~CP като входен
    DDRB &= ~_BV(PB7);
    PORTB &= ~_BV(PB7);

    // Стартiranе на програмируемия осцилатор
    osc_output_enable(oscoe);
    while (1) {

    }
}
```

фиг. 5.62: Примерна програма за настройване и стартиране на DDS системата

Използвайки описаните в тази точка функции, на фиг. 5.62 е представена примерна програма, която конфигурира DDS системата и стартира генерирането на сигнал. Първоначално пинът **CP** е конфигуриран като изходен, защото е необходим за управление на 16-битовия адресен брояч от микроконтролера. След това се конфигурира регистърът **MUX WORD** на DS1085 и осцилаторът се деактивира. Следва дефинирането на масив от данни, описващи желаната форма на генерирация сигнал. Важно е да се отбележи, че броят на точките в масива трябва да е степен на 2. Последното условие е необходимо, защото данните за формата на сигнала се записват няколкократно до запълване на SRAM паметта. Удобството на неколкократния повтарящ се запис е, че е необходимо само и единствено инкрементиране на адресния брояч в процеса на генериране на сигнал. След достигане на своята последна стойност (последния адрес) адресният брояч автоматично се нулира на следващия импулс по **CP** линията. Ако точките, описващи сигнала, се запишат еднократно в паметта без да я запълнят, в процеса на генериране би се изисквало периодично нулиране на адресния брояч, което би усложнило както и хардуера, така и управляващия софтуер на проектираното устройство. След успешен запис на паметта данните се подават на ЦАП-а, нулира се адресният брояч и пин **CP** на микроконтролера се настройва като входен, за да не влияе на действието на програмируемия осцилатор, свързан към същата проводяща линия. Описаното досега настройване е минималното и задължително необходимо за правилното действие на DDS системата. След изпълнение на тази конфигурация е възможно да се активира програмируемият осцилатор, стартирайки непрекъснатият процес на генериране на сигнал с произволна форма.

5.5.3 Настройване параметрите на генерирания сигнал

Микроконтролерът може да изменя параметрите на генерирания сигнал, настройвайки няколко от интегралните схеми, съставящи устройството.

Понеже програмируемият осцилатор управлява адресния брояч в процеса на генериране на произволен сигнал, честотата на последния зависи от честотата на трептение на изхода **OUT1** на DS1085. Както бе коментирано в т. 3.1.5, в интегралната схема е вграден източник на трептения, генериращ тактовата честота за двета изхода. Честотата на тактовите трептения се настройва посредством два регистра - **OFFSET BYTE** и **DAC WORD**, чиито формати са дадени съответно на фиг. 5.64 и на фиг. 5.65. Стойността на **OFFSET BYTE** оказва честотния диапазон (виж фиг. A.1), а стойността на **DAC WORD** - броят инкрементални стъпки (по 10kHz), добавени към минималната честота в избрания диапазон.

Следователно тактовата честота на вградения источник на трептения се определя по формулата [5.63], взета от каталожните данни⁽¹³⁾ на DS1085:

$$Frequency = \text{Min Frequency} + DAC \times \text{Step Size} \quad [5.63]$$

OFFSET BYTE (Address 0Eh)								
								MSB
X	X	X	O4	O3	O2	O1	O0	LSB
<i>X = Don't care.</i>								

фиг. 5.64: Формат на регистъра **OFFSET BYTE** на DS1085

DAC WORD (Address 08h)

DAC WORD (Address 08h)																
First Data Byte								LSB	d2	d1	d0	X	X	X	X	LSB
MSB	d9	d8	d7	d6	d5	d4	d3	LSB	d2	d1	d0	X	X	X	X	LSB
<i>X = Don't care.</i>																

фиг. 5.65: Формат на регистъра **DAC WORD** на DS1085

Честотата на резултатното трептение се разделя от делителя P1 (виж фиг. 3.1) на 1, 2, 4 или 8. Коефициентът на деление се определя от битовете **1M0** и **1M1** на регистъра **MUX WORD** (виж фиг. 5.58).

След това честотата на трептение може да се раздели допълнително и от програмируем делител в обхвата от 2 до 1025. Коефициентът на деление на програмирамия делител се задава със стойността на регистъра **DIV WORD**, чийто формат е показан на фиг. 5.66, като на фиг. A.17 е показано съответствието между стойността на регистъра и коефициента на деление.

DIV WORD (N) (Address 01h)

DIV WORD (N) (Address 01h)																
First Data Byte								LSB	Second Data Byte							
MSB	N9	N8	N7	N6	N5	N4	N3	LSB	N2	N1	N0	X	X	X	X	LSB
<i>X = Don't care.</i>																

фиг. 5.66: Формат на регистъра **DIV WORD** на DS1085

```
26 uint8_t osc_conf_freq(uint8_t address, uint16_t dac_word, uint8_t offset_byte);
27
28 uint8_t osc_conf_div(uint8_t address, uint16_t div_word);
```

фиг. 5.67: Прототипи на функциите „osc_conf_freq()“ и „osc_conf_div()“

файл: „include/DFG-firmware/programmable_oscillator_driver.h“

На фиг. 5.67 са дадени прототипите на функциите „*osc_conf_freq()*“ и „*osc_conf_div()*“. Първата се използва за задаване честотата на трептене на вградения осцилатор на DS1085 чрез конфигуриране на регистрите **OFFSET BYTE** и **DAC WORD**. Другата функция променя стойността на **DIV WORD**, и съответно задава коефициента на деление на програмируемия делител.

Честотата на генерирация сигнал зависи и от още един фактор - броят точки, използвани за описание на формата на сигнала. Например ако генератора на сигнали трябва да генерира сигнал, описан с 64 точки, с честота 1MHz, то програмируемият осцилатор трябва да генерира трептене с честота 64MHz на изхода си **OUT1**, защото за описание на един период от сигнала е необходимо да се обходят 64 клетки от SRAM паметта. Следователно честотата на генерирация сигнал може да се изчисли по формулата [5.68]:

$$f_{sig.} = \frac{f_{OUT1}}{res_{sig.}} \quad [5.68]$$

където:

$f_{sig.}$ - желаната честота на генерирация сигнал

f_{OUT1} - честота на трептене на изхода **OUT1** на DS1085

$res_{sig.}$ - резолюцията на генерирация сигнал, т.е. броят точки за описание на един

негов пълен период

Към генерирация сигнал може да се добави постоянно отместване в диапазона $-5V \div +5V$, настройвайки цифровия потенциометър MCP4551-103E (**U15** на фиг. 3.35) през I²C интерфейс. Тази интегрална схема съдържа няколко 10-битови регистри, запазени в енергозависима памет, с които се описва поведението ѝ. Манипулирането на тези регистри се осъществява с 4 команди - запис (**WRITE**), четене (**READ**), инкрементиране (**INCREMENT**), декрементиране (**DECREMENT**). На фиг. 5.69 е показан форматът на I²C съобщение при изпращане на команда за записване на нова стойност в избран регистър. Първият байт е контролният, който съдържа I²C адреса на интегралната схема. Със следващия байт се оказва първо адреса на манипулирания регистър и команда за манипулиране. Най-младшите два бита на втория байт представляват най-старшите битове на всеки 10-битов регистър (**D9** и **D8**). Важно е да се отбележи, че най-старшият бит (**D9**) не се използва и неговата стойност се пренебрегва от потенциометъра. В последния трети байт се оказват новите състояния на най-младшите 8 бита (**D7** \div **D0**) на манипулирания регистър.

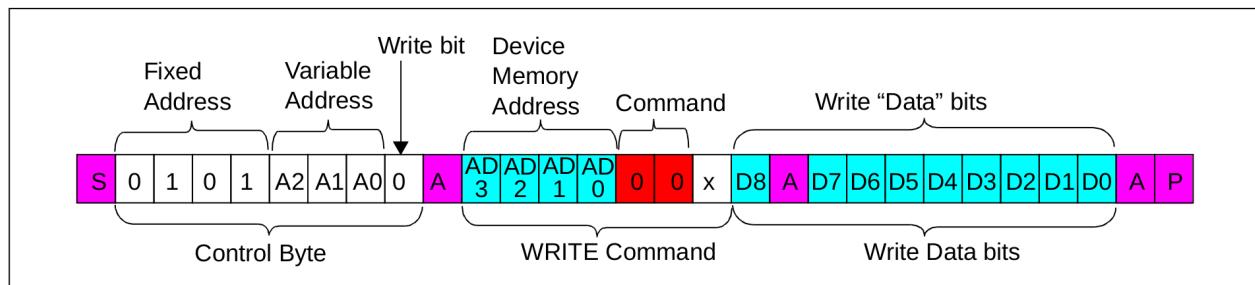


FIGURE 7-2: I^2C Write Sequence.

фиг. 5.69: Формат на съобщение за записваща команда към MCP4551

За промяна на приложеното постояннонотоково отместване на генеририания сигнал е достатъчно само да се запише нова стойност в регистъра **VOLATILE WIPER 0**⁽³⁴⁾ (с адрес 0_{16}) на MCP4551. Функцията „*set_dc_offset()*“, показана на фиг. 5.70, като параметър приема стойност за записване в регистъра **VOLATILE WIPER 0**. Максималната стойност на параметъра е ограничена до 100_{16} (256_{10}). След това се изгражда I^2C съобщение, включващо адреса и новата стойност на регистъра, по образца от фиг. 5.69. Накрая съобщението се изпраща през TWI интерфейса на ATmega2561. Макросите, започващи с „*MCP4551_*“, са дефинирани в хедър файла „*include/DFG-firmware/chips/MCP455X.h*“.

```

97 uint8_t set_dc_offset(uint16_t wiper_value)
98 {
99     uint8_t rc;
100    uint8_t pot_data[2];
101
102    wiper_value = (wiper_value & 0x100) ? 0x100 : wiper_value; // cap the wiper value at 100h
103    pot_data[0] = MCP4551_VOLATILE_WIPER_0 // address volatile wiper 0 register
104        | MCP4551_WRITE_CMD // write command
105        | ((wiper_value >> 8) & 0x01); // D8 bit
106    pot_data[1] = wiper_value & 0xFF; // D7-D0 bits
107    if ((rc = twi_write(U15_TWI_ADDRESS, pot_data, 2)) != 0) return rc;
108    return 0;
109 }
```

фиг. 5.70: Задаване на постояннонотоковото отместване на генеририания сигнал

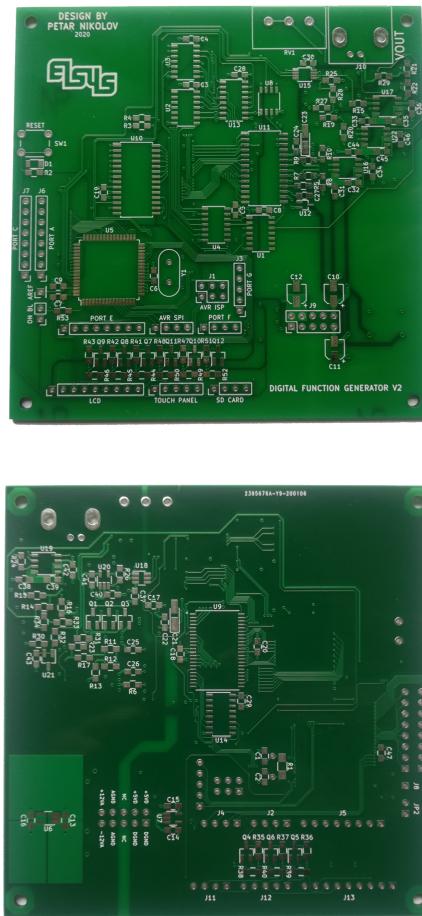
файл: „src/DFG-firmware/DDS_firmware.c“

ГЛАВА VI

Създаване на работоспособен модел на цифровия генератор на сигнали

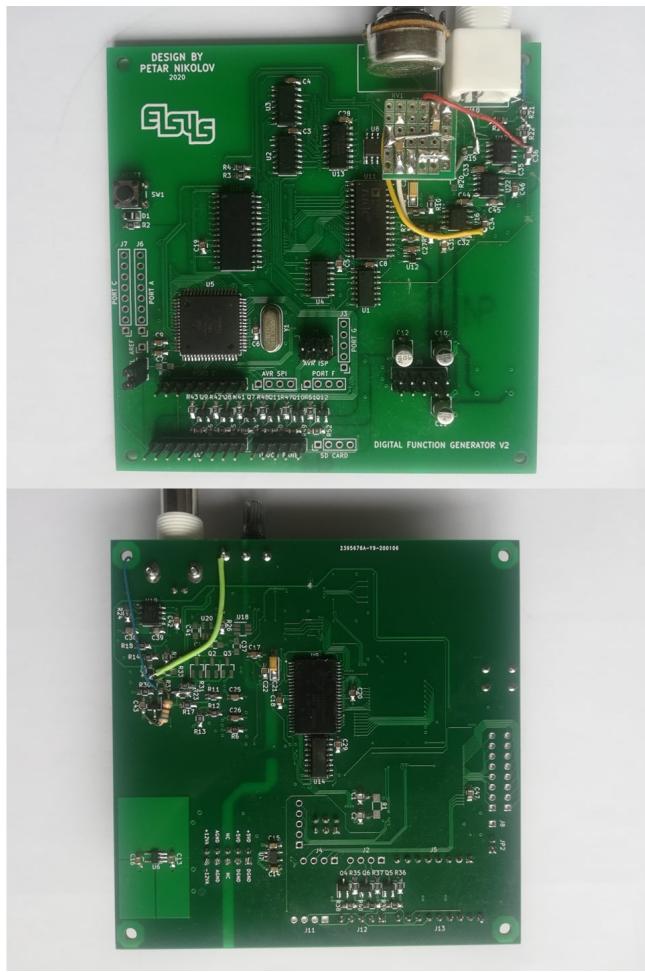
6.1 Оживяване на печатната платка на генератора на сигнали

За създаването на работоспособен модел на генератора са поръчани печатни платки от производителя JLCPCB⁽⁴³⁾. На фиг. 6.1 е представен екземпляр от получената пратка. Всички получени екземпляри са в отлично и работоспособно състояние.



фиг. 6.1:Печатна платка, произведена от JLCPCB, на генератора на сигнали

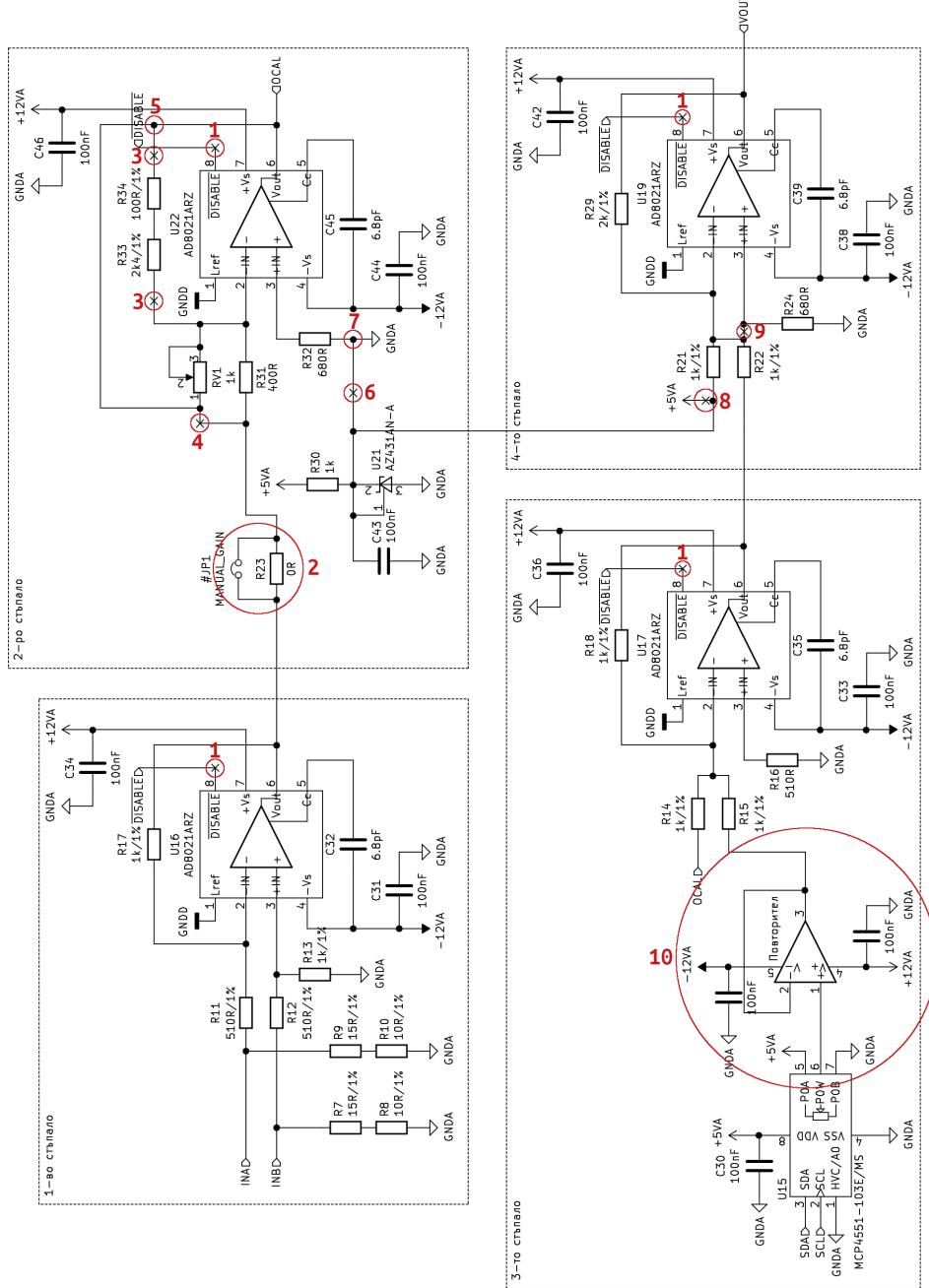
На фиг. 6.2 е представен екземпляр с налепени електронни компоненти. Важно е да се отбележи, че компонентите, нужни за електронно регулиране на амплитудата на генерираия сигнал, не са закрепени. Описаният в тази глава работоспособен модел използва предвидения ръчен метод за изменяне на амплитудата (виж т.3.2.6).



фиг. 6.2: Оживена печатна платка на цифровия генератор на сигнали

6.1.1 Модификации на печатната платка

Както се забелязва на фиг. 6.2, налепеният екземпляр има и допълнителни електрически връзки, реализирани с допълнителни проводници, и прекъснати проводящи писти. В процеса на оживяване на платката са направени модификации на принципната електрическа схема, по-специфично - на изходната аналогова схемотехника от фиг. 3.35. На фиг. 6.3 е показана модифицираната електрическа схема, като в червени кръгове с номера са оградени нанесените промени. На схемата не са показани електронните компоненти, необходими за електронното изменение на амплитудата на генерираия сигнал, защото не са залепени на работоспособната платка.



фиг. 6.3: Модифицирана принципна схема на блок „Изходната аналогова схемотехника“

Електрическите връзки между логическия инвертор **U1F** и входните изводи **DISABLE** на всички операционни усилватели AD8021 са прекъснати (означения „1“ на фиг. 6.3). Напрежението на изводите **DISABLE** спрямо **LOGIC REFERENCE** на всеки един от използваните операционни усилватели AD8021 експериментално е установено да е с приблизителна стойност 7V. Понеже интегралната схема 74HC04 работи с логически нива до +5V, след захранване на схемата е установено, че логическият елемент **U1F** е повреден трайно и непоправимо. Чрез прекъсване на връзките между изводите **DISABLE** и всякакви други проводящи писти се губи способността за „изключване“ - съществено намаляване на консумирания ток, на операционните усилватели.

Второто стъпало е изменено коренно спрямо първоначалната електрическа схема от фиг. 3.32, но запазва основното си предназначение - регулиране на амплитудата на генерираия сигнал. Тъй като на работоспособната платка се използва ръчно управление на амплитудата, джъмперът **JP1** е затворен, което се постига чрез окъсяване на изводите на **R23** (означение „2“ на фиг. 6.3). Резисторите **R33** и **R34** не са залепени за платката, което символично е показано на фиг. 6.3 чрез прекъсване на електрическите връзки (означения „3“). Проводящата писта, свързана към извода 1 на **RV1**, е срязана (означение „4“), като същият извод е свързан към изхода на усилвателя **U22** (означение „5“), използвайки допълнителен проводник. Резисторът **R31** е заменен с 4 паралелно свързани резистори със номинално съпротивление 1.6k Ω . Следователно съпротивлението на **R31** е 400 Ω . От всички имплементирани и описани промени досега следва, че коефициентът на усиливане на второто стъпало се изчислява по формулата [6.4]:

$$GAIN_2 = \frac{RV1}{R31} = \frac{0 \div 1000 \Omega}{400 \Omega} = 0 \div 2.5 \quad [6.4]$$

Дипломантът е допуснал и друга грешка в принципната схема на второто стъпало. В схемата от фиг. 3.35 опорното напрежение +2.5V е подадено на неинвертиращия вход на **U22** с цел добавяне на постояннотоково отместване от +2.5V към изходния сигнал на стъпалото. Но нормалната реакцията на всеки операционен усилвател е да изравни напрежението и на двета си входа. Следователно напрежението на инвертиращия вход също е +2.5V, което също ще бъде усилено от усилвателя. При липса на входен сигнал изходното напрежение на второто стъпало е:

$$V_{OUT2} = GAIN_2 \times V_{IN} = (0 \div 2.5) \times 2.5 V = 0 \div 6.25 V \quad [6.5]$$

Изчисленото по израза [6.5] добавено постояннотоково отместване зависи от коефициента на усилване, което не е желаното от проектанта поведение. Затова пистата между източника на опорно напрежение **U21** и резистора **R32** е прекъсната (означение „6“ на фиг. 6.3). Резисторът е свързан към аналоговата земя (означение „7“), а източникът - към инвертиращия вход на усилвателя **U19** (означение „8“), като връзката между резистора **R21** и напрежението +5V_A също е прекъсната. По този начин хем второто стъпало не добавя постоянно напрежение вместо +5V. Макар и да е променена принципната схема съществено, поведението на четиристъпалния усилвател все пак е по предназначението, замислено и описано в т.3.2.6. Единственият недостатък на новата схема е, че изходният сигнал на второто стъпало може да е както и с положително ниво спрямо **GNDA**, така и с отрицателно, понеже не се добавя положително постоянно напрежение към стъпалото. Следователно обратната връзка от изхода на второто стъпало към един от каналите (**PF3**) на вградения АЦП на микроконтролера ATmega2561 трябва да бъде прекъсната. В противен случай ако към микроконтролера се подаде отрицателно напрежение, той би бил трайно повреден.

Принципната схема на четиристъпалния усилвател съдържа и още един проблем. Дипломантът не е предвидил, че операционният усилвател **U17**, включен в схема „сумиращ усилвател“, инвертира входните си сигнали. Ако пъзгачът на цифровия потенциометър **U15** се постави в средна позиция, подавайки +2.5V като един от входните сигнали на схемата „сумиращ усилвател“, то при липса на генериран сигнал на изхода на **U17** се установява напрежение с големина -2.5V. Понеже последното четвърто стъпало от фиг. 3.35 представлява схема тип „изваждащ усилвател“, следва, че на изхода му ще се установи напрежението:

$$V_{OUT4} = (V_{OUT3} - 2.5V) \times GAIN_4 = (-2.5V - 2.5V) \times 2 = -10V \quad [6.6]$$

Желаното поведение на усилвателя е да установи постоянно напрежение от 0V при средно положение на пъзгача на **U15**, но според израза [6.6] това е невъзможно. Затова на платката пистата между **R22** и **R24** е срязана, като **R22** се свързва към инвертиращия вход на операционния усилвател **U19** (означение „9“ на фиг. 6.3). Също така на мястото на резистора **R24** е поставен нов резистор с номинално съпротивление 680Ω, който симетрира входовете на **U19**. Със споменатите промени последното стъпало вече представлява схема тип „сумиращ усилвател“, при която изходното напрежение на усилвателя **U19** е:

$$V_{OUT4} = -(V_{OUT3} + 2.5V) \times GAIN_4 = -((0 \div -5V) + 2.5V) \times GAIN_4 = \pm 5V \quad [6.7]$$

По този начин се постига първоначално предвидената способност за добавяне на постояннотоково отместване в диапазона $\pm 5V$.

Третото стъпало има още един съществен проблем. Един от входните сигнали за сумирация усилвател представлява постояннотоковото ниво, зададено с цифровия потенциометър **U15**. Плъзгача на потенциометъра е свързан към инвертиращия вход на **U17** през резистора **R15**. Описаният начин на свързване е показан на фиг. 6.10. Оказва се, че резисторът **R15** е свързан към виртуалната „0“ на операционния усилвател, т.е. при анализиране на схемата може да се приеме, че е свързан към земята **GNDA**. Следователно може да се приеме, че резисторът **R15** е свързан паралелно със съпротивлението **R_{WB}** на потенциометъра. Ако плъзгачът (W) е поставен в средно положение, резултатното съпротивление е:

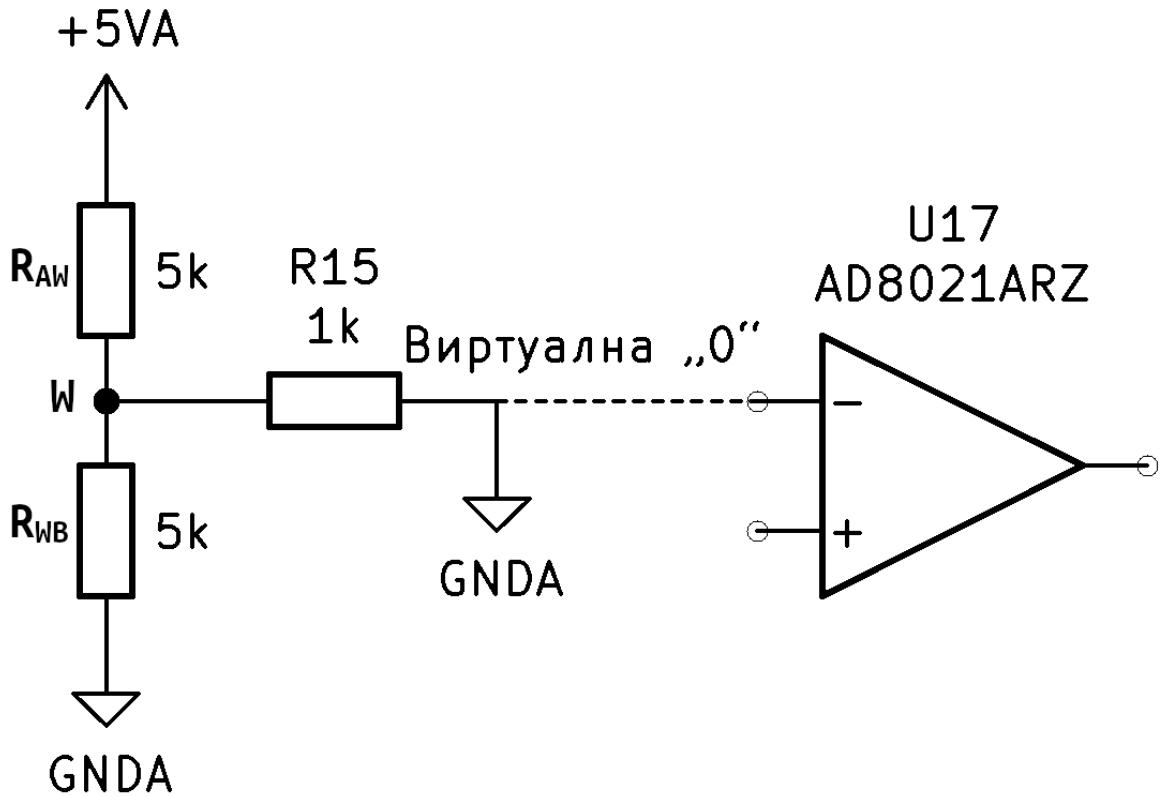
$$R_{R_{WB}\parallel R15} = 5k\parallel 1k = \frac{5 \times 1}{5+1} \approx 833\Omega \quad [6.8]$$

От израза ??? следва, че постояннотоково ниво на плъзгача е:

$$V_w = \frac{R_{R_{WB}\parallel R15}}{R_{R_{WB}\parallel R15} + R_{AW}} \times 5V = \frac{833\Omega}{833\Omega + 5k\Omega} \times 5V \approx 0.71V \quad [6.9]$$

Дипломантът опитно е потвърдил изчислението [6.9] чрез извършени измервания. Следователно поведението на третото стъпало е изключително непрецизно и нездоволяващо. Изразът е пример, че при очаквано добавено постояннотоково ниво от $+2.5V$ се установява ниво $+0.71V$.

На оживената платка този проблем е премахнат чрез добавяне на операционен усилвател (означение „10“ на фиг. 6.3) между цифровия потенциометър и входа на „сумирация усилвател“. Добавеният усилвател е свързан в схема тип „повторител на напрежение“, т.е. представлява неинвертиращо буферно стъпало с коефициент на усиливане единица. По този начин се избягват всякакви непредвидени влияния върху нормалното действие на **U15** от страна на останалата аналогова схемотехника. За буферното стъпало може да се използва всякакъв операционен усилвател, като единственото условие е последният да може да се захрани с двойното напрежение $\pm 12V$. В процеса на изграждане на работоспособна платка е използван операционният усилвател LM358P⁽⁴⁵⁾, понеже е широкоразпространен и евтин компонент.



фиг. 6.10: Анализ на принципното свързване на цифровия потенциометър **U15** с третото стъпало

6.1.2 Други проблеми на проектираното устройство

Проектираното устройство има и един значителен проблем в принципната схема на цифровата си част. Както бе споменато в т.3.2.5.2, 16-битовият брояч е съставен от два 12-битови броячи (74HC4040), свързани помежду си с логически елементи от серията 74HC (виж фиг. 3.34). Проектираният по този начин схема е предпоставка за явлението „състезание на сигнали“. При инкрементиране на стойността на **U13** от 4094 (предпоследната за **U13**) на 4095 (максималната за **U13**) стойността на **U14** се увеличава. Следователно стойността на **U14** се увеличава два пъти при две поредни отброявания, т.е. двойно повече от предназначеното.

Този проблем е решен изцяло чрез модифициране на управляващия код на устройството. Вторият брояч (**U14**) се държи с постоянно нулирана стойност, като линията **CMR2** непрекъснато се държи във високо логическо ниво. Недостатъкът на това решение е, че не може да се оползотвори пълният капацитет на SRAM паметта на DDS системата - възможно е да се адресират само първите 4096 клетки вместо всичките 65,536.

6.2 Зареждане на управляващия софтуер

За инсталиране на управляващия софтуер на работоспособната платка е необходимо програмата да се компилира в поредица от машинни инструкции, които да се запишат в програмната памет на микроконтролера ATmega2561.

6.2.1 Компилиране на управляващата програма

Интегрираната програмна среда MPLAB X IDE v5.25, с която е разработена управляващата програма, използва конзолната програма **make** за компилиране на програмния код на разработения софтуер. Следователно е възможно проекта да се компилира без намесата на програмната среда, използвайки конзолата на операционната система.

```
$ make build
```

фиг. 6.11: Компилиране на управляващата програма

След изпълняване на командалата се генерира файл с машинен код за записване в програмната памет на ATmega2561, като файлът е „*dist/default/production/DFGv2-Firmware.X.production.hex*“. Последният съдържа машинните инструкции на управляващата програма за ATmega2561, записани в шестнайсетичен формат.

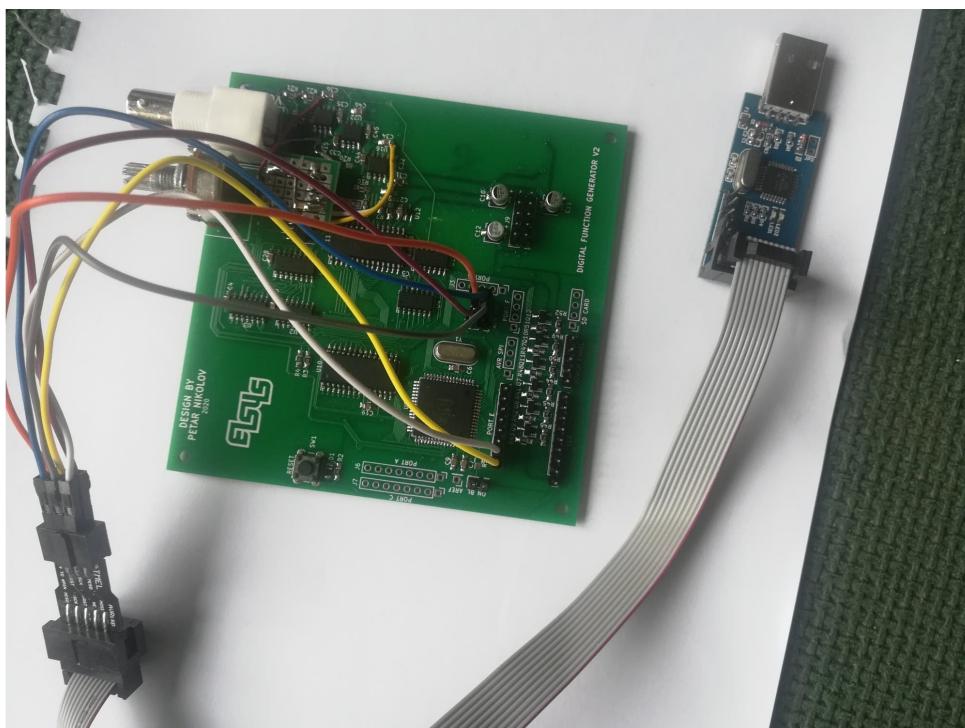
6.2.2 Програмиране на микроконтролера ATmega2561

За да се програмира микроконтролерът ATmega2561, е необходим подходящ програматор. На фиг. 6.12 е показан програматорът USBasp⁽⁴⁴⁾. Той се включва към машината, която съдържа файла с компилирания машинен код, посредством USB интерфейс. От друга страна, програматорът зарежда компилираната програма в микроконтролера, използвайки SPI интерфейса му.



фиг. 6.12: USBasp програматор за AVR микроконтролери

В т.3.2.3 е казано, че програматорът за микроконтролера трябва да се свърже към 6-пиновия съединител **J1** от фиг. 3.32. Самият съединител се връзва към захранването и към SPI порта на ATmega2561. На фиг. А.19 е представена таблица от каталожните данни на ATmega2561, която показва към кои изводи трябва да се свърже програматорът при програмиране на микроконтролера чрез сериен интерфейс. Оказва се, че програматорът не трябва да е свързан към пиновете **PB2 (MOSI)** и **PB3 (MISO)**, както е на схемата от фиг. 3.32, а към пиновете **PE0 (PDI)** и **PE1 (PDO)**. На фиг. 6.13 е показано как програматорът е свързан към оживената платка.



фиг. 6.13: Свързване на програматора към работоспособната печатна платка

След свързването си към микроконтролера програматорът трябва да се включи в един от USB портовете на машина, съдържаща файла с машинния код за качване на микроконтролера. После програмната памет на ATmega2561 се програмира, използвайки отново конзолната програма **make**, както е показано на фиг. 6.14. Задължително е машината, изпращаща файла с машинния код, да има инсталирана конзолната програма **avrdude**, която установява комуникация с и контролира USBasp програматора.

```
$ make flash
```

фиг. 6.14: Записване на управляващата програма в програмната памет на ATmega2561

Освен зареждането на управляващия код във флаш паметта на микроконтролера, програматорът трябва да настрои и **fuse** байтовете му. ATmega2561 съдържа 3 такива байта - **low**, **high**, **extended**, всеки от които конфигурира различни хардуерни параметри на интегралната схема, които не могат да се манипулират от управляващия софтуер. Тези 3 байта може да бъдат програмирани с команда, показана на фиг. 6.15:

```
$ make fuse
```

фиг. 6.15: Програмиране на конфигурационните байтове на ATmega2561

Стойностите, които се зареждат в fuse байтовете са следните:

- **low** (l): FF₁₆
- **high** (h): 99₁₆ (стойност по подразбиране)
- **extended** (f): FF₁₆ (стойност по подразбиране)

Байтовете **high** и **extended** остават непроменени спрямо фабричните настройки на ATmega2561. От друга страна, в байта **low** е заредена стойността FF₁₆ (т.е. всички битове са програмирани), с което се оказва на микроконтролера за източник на тактов сигнал да използва външен кристален осцилатор с честота на трептене $\geq 8\text{MHz}$. Тази настройка е задължителна за използването на кристалния осцилатор **Y1** от микроконтролера.

6.3 Изследване на изградения работоспособен модел

6.3.1 Опитна постановка

На фиг. 6.16 е показана опитната постановка, която се използва за изследване на проектираното устройство. В центъра на снимката е работоспособната платка на цифровия генератор на сигнали. Последният се захранва от двойно захранване (вляво на снимката). Към BNC изхода на устройството е включен осцилоскоп, с който се изследва генерирания от устройството сигнал.

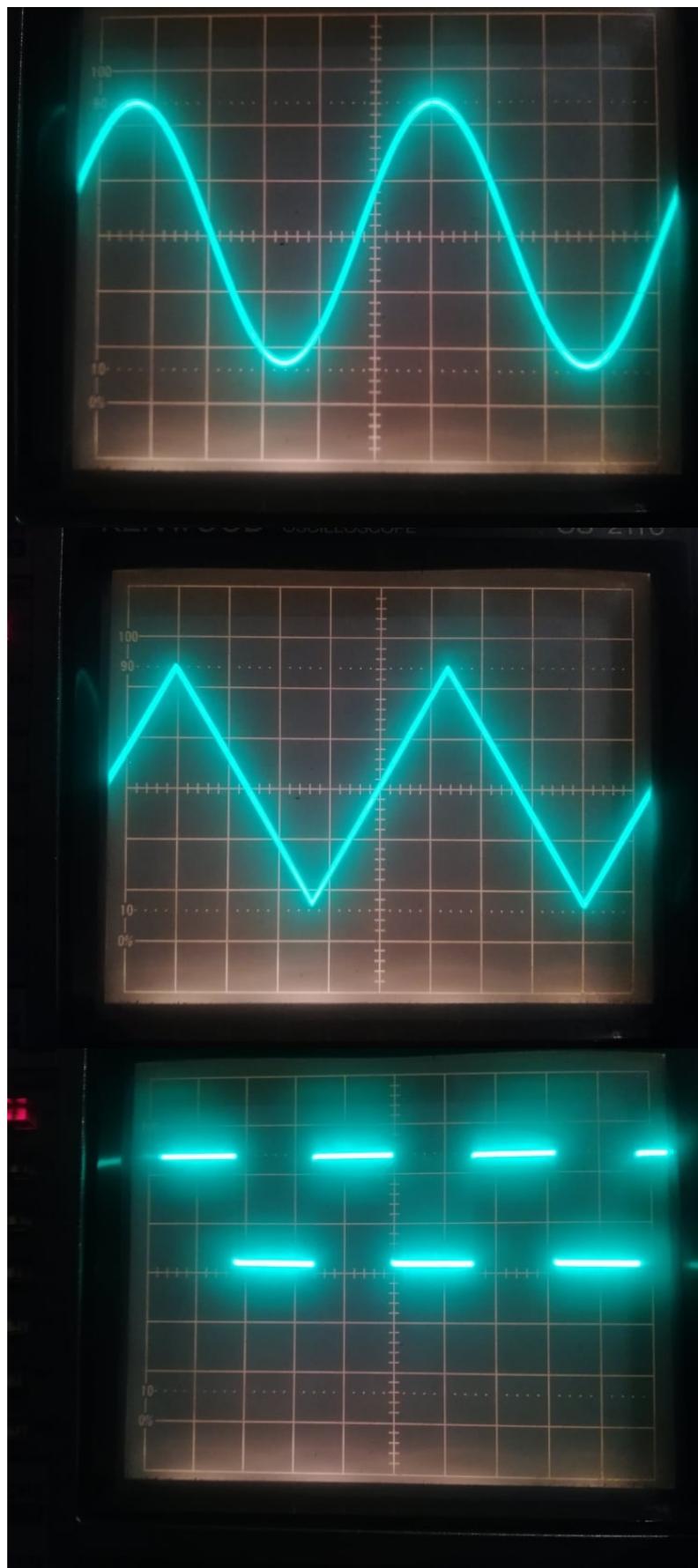


фиг. 6.16: Опитна постановка, използвана за изследване на проектираното устройство

6.3.2 Резултати от изследванията

Първата характеристика, която е изследвана от проектанта, е възможността на генератора да възпроизведе електрически сигнали с произволна форма. На фиг. 6.17 са показани снимки на екрана на осцилоскопа, от които се забелязва, че генераторът може да възпроизведе сигнали със синусоидална, триъгълна и правоъгълна форма. Синусоидалният и триъгълният сигнал са описани с 1024 стойности/период и имат амплитуда 10V от връх до връх, а правоъгълният - с 2 стойности/период и 5V от връх до връх.

Тъй като различната форма на показаните сигнали е постигната само с промяна на стойностите, записани в паметта на DDS системата, може да се направи изводът, че генераторът може да възпроизведе електрически сигнали с произволна форма.

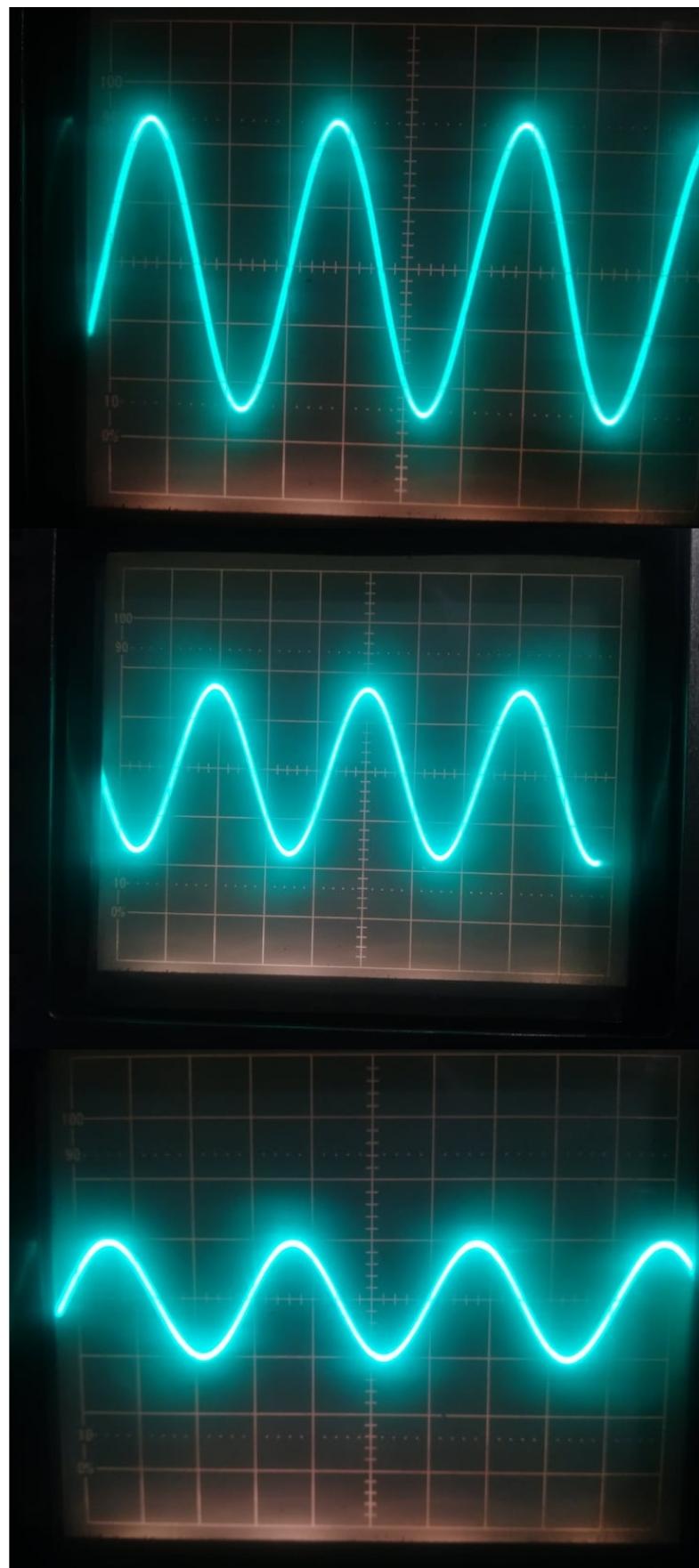


фиг. 6.17: Генериирани сигнали със синусоидална (горе), триъгълна (по средата) и правоъгълна (долу) форма

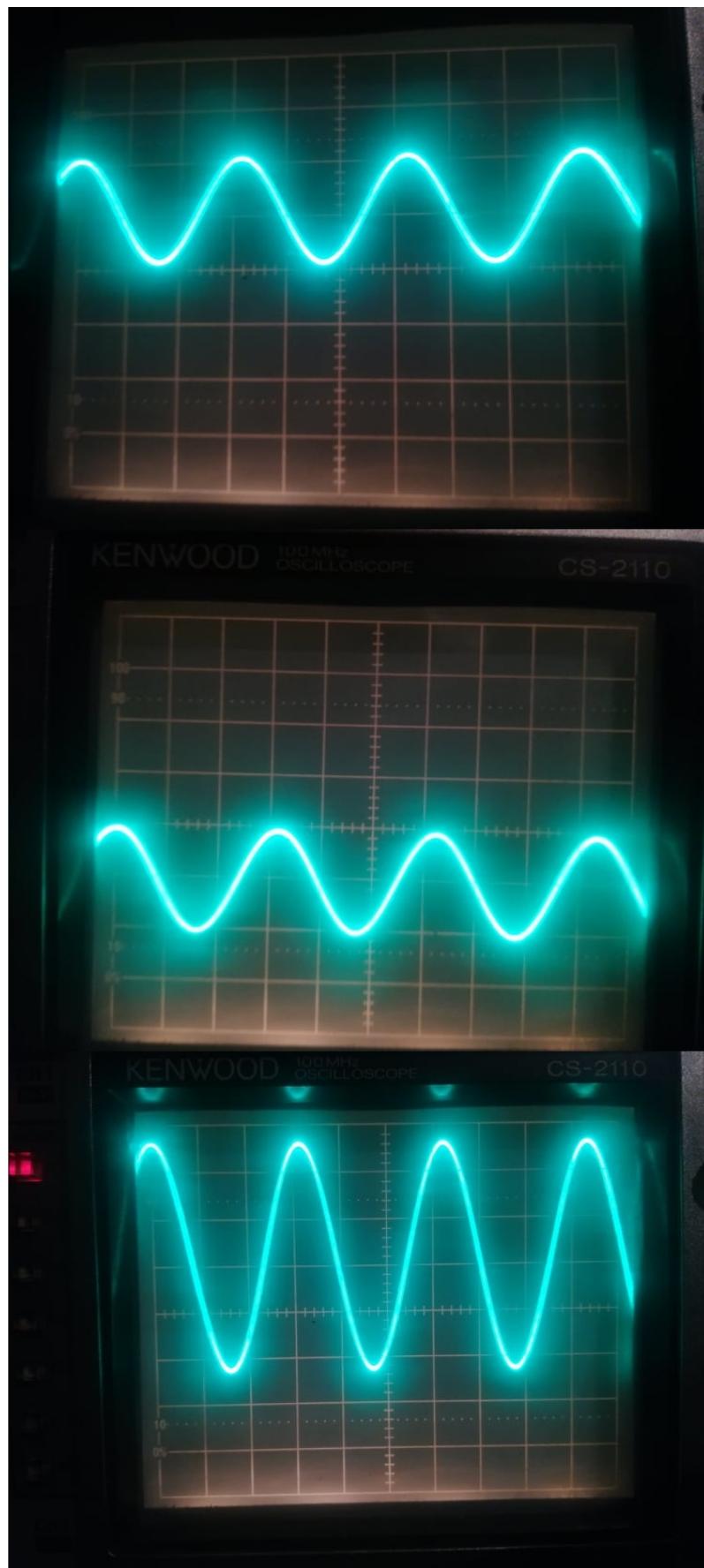
На фиг. 6.18 са показани снимки на един и същ синусоидален сигнал (с еднаква честота и форма), като единствената им разлика е в амплитудата. Амплитудата на сигнала се регулира посредством механичния потенциометър **RV1**. Забележително е, че не са налични каквито и да са нелинейни изкривявания на генерирания сигнал.

На фиг. 6.19 са показани снимки на един и същ синусоидален сигнал с различни приложени постояннотокови отмествания, което се постига чрез настройване на цифровия потенциометър **U15**. Всички сигнали са с амплитуда 10V от връх до връх, като към първия е добавено постояннотоково ниво +5V, към втория - -5V, към третия - +2V. Тук отново е забележително, че няма налични нелинейни изкривявания на генерираните сигнали.

От направените изследвания може да се заключи, че успешно е проектиран цифров генератор на сигнали с произволна форма, с максимална амплитуда $\pm 5V$ и с възможност за добавяне на постояннотоково отместване в диапазона $-5V \div +5V$.



фиг. 6.18: Един и същ синусоидален сигнал с изменена амплитуда - 10V p-p (горе),
6.6V p-p (по средата), 4V p-p (долу)



фиг. 6.19: Един и същ синусоидален сигнал с различно постояннотоково отместавне - +5V (горе), -5V (по средата), +2V (долу)

6.4 Икономическият аспект

На таблица 6.22 са представени нагледно разходите (в български лева) за всички използвани електронни елементи, като някои от последните са поръчани от Comet Electronics⁽⁴⁶⁾, някои от AliExpress⁽⁴⁷⁾. Елементите от AliExpress нямат добавено ДДС, понеже пристигат от Китай и не са задържани на митницата. Въпреки това за някои от поръчаните китайски компоненти е заплатена сума за доставка. На последния ред от таблицата са дадени и общата сума за елементите без ДДС и разходи за доставка, и заплатената обща сума (с ДДС и разходи за доставка).

На предпоследния ред на същата таблица са дадени и разходите за печатните платки, произведени от JLCPCB⁽⁴³⁾. Доставката от Китай е направена през DHL Express, поради което самата сума за доставката е сравнително голяма. Важно е да се отбележи, че поръчката с печатните платки е задържана на митницата, поради което дипломантът е заплатил и допълнителни административни разходи, представени в таблица 6.20. В таблица 6.21 са включени и споменатите административни разходи, и разходите за електронните елементи, показвайки общата заплатена сума от дипломанта за дипломния проект.

Описание	Сума
Издаване на еднократен ЕОРИ номер	25.00 лв.
Нотариално заверено пълномощно, удостоверяващо DHL Express да представя поръчвателя пред митницата	6.00 лв.
Такса митница (20%)	18.83 лв.
ОБЩО	49.83 лв.

таблица 6.20: Административни разходи за поръчката с печатните платки

Разход	Сума
Елементи	252.79 лв.
Административни	49.83 лв.
ОБЩО	302.62 лв.

таблица 6.21: Общи разходи за дипломния проект

Елемент(и)	Бройка елементи в партида	Количество поръчани партиди	Цена на партида	Стойност	ДДС	Доставка	Общо
-	-	-	BGN	BGN	%	BGN	BGN
C0805 22pF	100	1	1.68 лв.	1.68 лв.	0.00%	0.00 лв.	1.68 лв.
C0805 100nF	100	1	1.68 лв.	1.68 лв.	0.00%	0.00 лв.	1.68 лв.
AI CP 10uF/16V	20	1	1.60 лв.	1.60 лв.	0.00%	0.94 лв.	2.54 лв.
C0805 2.2uF	1	10	0.035 лв.	0.35 лв.	20.00%	0.00 лв.	0.42 лв.
C0805 1uF	1	10	0.033 лв.	0.33 лв.	20.00%	0.00 лв.	0.42 лв.
CP Tant 10uF	10	1	1.78 лв.	1.78 лв.	0.00%	0.00 лв.	1.78 лв.
C0805 6.8pF	1	10	0.012 лв.	0.12 лв.	20.00%	0.00 лв.	0.14 лв.
R0805 10kΩ,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 2kΩ,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 1kΩ,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 1.6kΩ,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 910Ω,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 680Ω,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 560Ω,5%	100	1	1.05 лв.	1.05 лв.	0.00%	0.00 лв.	1.05 лв.
R0805 2kΩ,1%	1	10	0.041 лв.	0.41 лв.	20.00%	0.00 лв.	0.49 лв.
R0805 1kΩ,1%	1	10	0.041 лв.	0.41 лв.	20.00%	0.00 лв.	0.49 лв.
R0805 15Ω,1%	1	10	0.051 лв.	0.51 лв.	20.00%	0.00 лв.	0.61 лв.
R0805 10Ω,1%	1	10	0.053 лв.	0.53 лв.	20.00%	0.00 лв.	0.64 лв.
R0805 100Ω,1%	1	10	0.062 лв.	0.62 лв.	20.00%	0.00 лв.	0.74 лв.
R0805 510Ω,1%	1	10	0.041 лв.	0.41 лв.	20.00%	0.00 лв.	0.49 лв.
R0805 2.4kΩ,1%	1	10	0.062 лв.	0.62 лв.	20.00%	0.00 лв.	0.74 лв.
R0805 470kΩ,5%	1	10	0.031 лв.	0.31 лв.	20.00%	0.00 лв.	0.37 лв.
Лин. пот. 1k	1	1	0.67 лв.	0.67 лв.	20.00%	0.00 лв.	0.80 лв.
CD1206-SO1575	1	5	0.28 лв.	1.40 лв.	20.00%	0.00 лв.	1.68 лв.
MMBF5485	1	10	0.14 лв.	1.40 лв.	20.00%	0.00 лв.	1.68 лв.
AP2310GN	1	15	0.19 лв.	2.85 лв.	20.00%	0.00 лв.	3.42 лв.
BNC	10	1	3.74 лв.	3.74 лв.	0.00%	0.50 лв.	4.24 лв.
74HC04D	1	1	0.17 лв.	0.17 лв.	20.00%	0.00 лв.	0.20 лв.
74HC08D	1	2	0.17 лв.	0.34 лв.	20.00%	0.00 лв.	0.41 лв.
74HC30D	1	1	0.21 лв.	0.21 лв.	20.00%	0.00 лв.	0.25 лв.
74HC4040D	1	2	0.28 лв.	0.56 лв.	20.00%	0.00 лв.	0.67 лв.
ATmega2561-16AU	1	1	22.02 лв.	22.02 лв.	20.00%	0.00 лв.	26.42 лв.
TC1017-3.3VCTTR	1	1	0.76 лв.	0.76 лв.	20.00%	0.00 лв.	0.91 лв.
ADP7142AUJZ-5.0	1	1	5.93 лв.	5.93 лв.	20.00%	0.00 лв.	7.12 лв.
DS1085Z-10+	1	1	15.97 лв.	15.97 лв.	20.00%	0.00 лв.	19.16 лв.
CY7C1021D-10ZSXI	1	1	5.72 лв.	5.72 лв.	20.00%	0.00 лв.	6.86 лв.
MCP23017-E/SO	1	1	2.22 лв.	2.22 лв.	20.00%	0.00 лв.	2.66 лв.
AD9762ARZ	1	1	28.36 лв.	28.36 лв.	20.00%	0.00 лв.	34.03 лв.
TS4061AICT-1.25	1	1	1.97 лв.	1.97 лв.	20.00%	0.00 лв.	2.36 лв.
MCP4551-103E/MS	1	1	1.70 лв.	1.70 лв.	20.00%	0.00 лв.	2.04 лв.
AD8021ARZ	1	5	3.34 лв.	16.70 лв.	20.00%	0.00 лв.	20.04 лв.
MCP4716A0T-E/CH	1	1	1.56 лв.	1.56 лв.	20.00%	0.00 лв.	1.87 лв.
MIC6211YM5-TR	1	1	0.59 лв.	0.59 лв.	20.00%	0.00 лв.	0.71 лв.
AZ431AN-A	1	1	0.16 лв.	0.16 лв.	20.00%	0.00 лв.	0.19 лв.
Кварц HC49-S 16MHz	1	1	0.29 лв.	0.29 лв.	20.00%	0.00 лв.	0.35 лв.
PCB	5	1	49.27 лв.	49.27 лв.	0.00%	44.87 лв.	94.14 лв.
ОБЩО				183.29 лв.			252.79 лв.

таблица 6.22: Разходи за хардуерните компоненти на генератора на сигнали

Заключение

Цифровите генератори на сигнали са изключително полезни за всеки инженер лабораторни устройства. Въпреки това настоящата дипломна работа доказва, че вътрешната структура на подобно устройство е изключително сложна, като проектирането и оживяването ѝ е времеемко начинание, свързано с решаването на сложни проблеми.

С настоящата дипломна работа успешно е проектиран и създаден прототипен генератор на сигнали с произволна форма. Но изграденият работоспособен модел има и много проблеми и в принципната електрическа схема, и в проектираната печатна платка, и в управляващия софтуер. Следователно в бъдеще прототипът може да бъде развит допълнително в много технически аспекта, някои от които са:

- Аналоговата част на проектирания генератор е редно да се преработи основно и оптимизира.
- Макар че при полза на преобразуватели (ЦАП или АЦП) се постигат оптимални резултати с четиристойна платка, при която цифровата и аналоговата заземяващи плохи са разделени и разположени на отделен слой, е възможно подобни резултати да се постигнат и с двуслойна платка с една заземяваща плоча, използвайки техниката „partitioning“⁽⁶⁾. Използването на двуслойна платка вместо четиристойна значително намалява разходите на което и да е проектирано електронно устройство.
- Въпреки че драйверният софтуер за тъчскрийн модула е напълно функционален, потребителската графична среда може да бъде направена безкрайно сложна според нуждите на потребителя.

От изследвания икономически аспект в т.6.4 може да се заключи, че разходите за изграждане на работоспособен модел на генератора са високи дори и ако се пренебрегнат административните разходи. Бюджетните комерсиални устройства са с малко по-висока цена, но предлагат много по-голям и много по-добре оптимизиран набор от функционалности. Необходимо е хардуера на проектирания генератор да бъде преразгледан и оптимизиран за постигане на „минималистичен“ вариант, конкурентен с наличните пазарни опции.

Все пак е важно да се отбележи, че сблъскването и решаването на сложните проблеми, възникнали при проектиране на подобно устройство, е изключително ефективен начин за натрупване на практически опит в сферата на електрониката и програмирането на вградени микросистеми.

Използвани източници

(1) SDG800 series datasheet,

http://siglentna.com/wp-content/uploads/dlm_uploads/2017/10/SDG800_DataSheet_DS02008-E02D.pdf

(2) SDG6000X series datasheet,

http://siglentna.com/wp-content/uploads/dlm_uploads/2018/04/SDG6000X_DataSheet_DS0206X-E02A-1.pdf

(3) MT-085, <https://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf>, 2008

(4) Eva Murphy и Colm Slattery, "Ask The Application Engineer—33: All About Direct Digital Synthesis", <https://www.analog.com/media/en/analog-dialogue/volume-38/number-3/articles/all-about-direct-digital-synthesis.pdf>, 2004

(5) Sanjay Pithadia и Shridhar More, "Grounding in mixed-signal systems demystified, Part 1", <http://www.ti.com/lit/an/slyt499/slyt499.pdf>, 2013

(6) Sanjay Pithadia и Shridhar More, "Grounding in mixed-signal systems demystified, Part 2", <http://www.ti.com/lit/an/slyt512/slyt512.pdf>, 2013

(7) Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet,

https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

(8) CY7C1021D datasheet, <https://www.cypress.com/file/42721/download>

(9) AD9764 datasheet, <https://www.analog.com/media/en/technical-documentation/data-sheets/AD9764.pdf>

(10) AD9762 datasheet, <https://www.analog.com/media/en/technical-documentation/data-sheets/AD9762.pdf>

(11) AD9708 datasheet, <https://www.analog.com/media/en/technical-documentation/data-sheets/AD9708.pdf>

(12) 74HC4040 datasheet, https://assets.nexperia.com/documents/data-sheet/74HC_HCT4040.pdf

(13) DS1085 datasheet, <https://datasheets.maximintegrated.com/en/ds/DS1085.pdf>

(14) MCP23017 datasheet, <http://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf>

(15) AD8021 datasheet, <https://www.analog.com/media/en/technical-documentation/data-sheets/AD8021.pdf>

(16) TC1017 datasheet, <http://ww1.microchip.com/downloads/en/DeviceDoc/21813F.pdf>

- (17) ADP7142 datasheet, <https://www.analog.com/media/en/technical-documentation/data-sheets/ADP7142.pdf>
- (18) 74HC08 datasheet, https://assets.nexperia.com/documents/data-sheet/74HC_HCT08.pdf
- (19) 74HC30 datasheet, https://assets.nexperia.com/documents/data-sheet/74HC_HCT30.pdf
- (20) 74HC04 datasheet, https://assets.nexperia.com/documents/data-sheet/74HC_HCT04.pdf
- (21) StackExchange, Electrical Engineering, „What is the function of this diode in arduino uno(between +5V and reset)“, <https://electronics.stackexchange.com/questions/261921/what-is-the-function-of-this-diode-in-arduino-unobetween-5v-and-reset>
- (22) Rajan Arora, „I2C Bus Pullup Resistor Calculation“,
<http://www.ti.com/lit/an/slva689/slva689.pdf>, 2015
- (23) UM10204, „I²C-bus specification and user manual“, <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>, 2014
- (24) 74HC4040 datasheet, https://assets.nexperia.com/documents/data-sheet/74HC_HCT4040.pdf
- (25) ILI9341 datasheet, <https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf>
- (26) XPT2046 datasheet, <https://ldm-systems.ru/f/doc/catalog/HY-TFT-2,8/XPT2046.pdf>
- (27) Kingston Technology, microSDXC memory card specsheet,
https://www.kingston.com/datasheets/SDCIT-specsheet-64gb_us.pdf
- (28) Атанас Шишков, Полупроводникова техника / Част втора / Усилватели и интегрални схеми, Техника, София, 1981
- (29) AZ431AN-A datasheet, <https://www.diodes.com/assets/Datasheets/AZ431A.pdf>
- (30) AN105, FETs As Voltage-Controlled Resistors, <https://www.vishay.com/docs/70598/70598.pdf>
- (31) MMBF5485 datasheet, <https://www.onsemi.com/pub/Collateral/MMBF5486-D.pdf>
- (32) MCP4716 datasheet, <https://ww1.microchip.com/downloads/en/DeviceDoc/22272C.pdf>
- (33) MIC6211 datasheet, <http://ww1.microchip.com/downloads/en/DeviceDoc/mic6211.pdf>
- (34) MCP4551 datasheet, <http://ww1.microchip.com/downloads/en/devicedoc/ds-22096a.pdf>
- (35) BNC Connector 3D model, <https://grabcad.com/library/bnc-connector-5>
- (36) Alps RK163 3D model, <https://grabcad.com/library/potentiometer-8>
- (37) MPLAB X IDE, Microchip, <https://www.microchip.com/mplab/mplab-x-ide>
- (38) 3.2inch SPI Module ILI9341 SKU:MSP3218, LCDWIKI,
http://www.lcdwiki.com/3.2inch_SPI_Module_ILI9341_SKU:MSP3218
- (39) AVRFREAKS forum, Boolean Type, <https://www.avrfreaks.net/forum/boolean-type>
- (40) LCDWIKI, 3.2inch SPI Module MSP3218 User Manual EN,
http://www.lcdwiki.com/res/MSP3218/3.2inch_SPI_Module_MSP3218_User_Manual_EN.pdf
- (41) GitHub, dhepper/font8x8, https://github.com/dhepper/font8x8/blob/master/font8x8_basic.h

- (42) avr-libc, Data in Program Space, <https://www.nongnu.org/avr-libc/user-manual/pgmspace.html>
- (43) JLCPCB, Уебсайт, <https://jlpcb.com/>
- (44) USBasp, Уебсайт, <https://www.fischl.de/usbasp/>
- (45) LM358P datasheet, <https://www.ti.com/lit/ds/symlink/lm358.pdf>
- (46) Comet Electronics, Уебсайт, <https://www.comet.bg/>
- (47) AliExpress, Уебсайт, <https://www.aliexpress.com/>

Приложение

А Допълнителни графични и снимкови материали

DS1085

Table 6. FREQUENCY vs. OFFSET

OFFSET	DS1085Z-10 FREQUENCY RANGE	DS1085Z-25 FREQUENCY RANGE	DS1085Z-50 FREQUENCY RANGE
OS - 10	—	—	—
OS - 9	—	—	—
OS - 8	—	—	—
OS - 7	—	—	—
OS - 6	61.4 to 71.6	51.2 to 76.8	38.4 to 89.6
OS - 5	66.5 to 76.8	57.6 to 83.2	44.8 to 96.0
OS - 4	71.6 to 81.9	64.0 to 89.6	51.2 to 102.4
OS - 3	76.7 to 87.0	70.4 to 96.0	57.6 to 108.8
OS - 2	81.9 to 92.1	76.8 to 102.4	64.0 to 115.2
OS - 1	87.0 to 97.2	83.2 to 108.8	70.4 to 121.6
OS*	92.1 to 102.3	89.6 to 115.2	76.8 to 128.0
OS + 1	97.2 to 107.5	96.0 to 121.6	83.2 to 134.4
OS + 2	102.3 to 112.6	102.4 to 128.0	89.6 to 140.8
OS + 3	107.5 to 117.7	108.8 to 134.4	96.0 to 147.2
OS + 4	112.6 to 122.8	115.2 to 140.8	102.4 to 153.6
OS + 5	117.7 to 127.9	121.6 to 147.2	108.8 to 160.0
OS + 6	122.8 to 133.1	128.0 to 153.6	115.2 to 166.4

*OS is the OFFSET default setting. OS is the integer value of the five MSBs of RANGE register.

These ranges include values outside the oscillator range of 66MHz to 133MHz. When using these ranges, values of DAC must be chosen to keep the oscillator within range. Correct operation of the device is not guaranteed outside the range 66MHz to 133MHz.

фиг. А.1: Таблица 6 от каталожните данни⁽¹³⁾ на DS1085

Table 3. DEVICE MODE USING OUT1

PDN1 (BIT)	CTRL1 (PIN)	CTRL1 FUNCTION	OUT1 (PIN)	DEVICE MODE
0	0	Output Enable	OUT CLK	Active*
0	1		High-Z	
1	0	Power-Down	OUT CLK	Active
1	1		High-Z	Power-Down

*Factory default setting.

фиг. А.2: Таблица 3 от каталожните данни⁽¹³⁾ на DS1085

Table 2. DEVICE MODE USING OUT0

EN0 (BIT)	SEL0 (BIT)	PDN0 (BIT)	CTRL0 (PIN)	OUT0 (PIN)	CTRL0 FUNCTION	DEVICE MODE
0	0	0	1	High-Z	Power-Down*	Power-Down***
			0	High-Z		Active
0	1	0	1	MCLK/M	Mux Select	Active
			0	MCLK		
1	0	0	1	High-Z	Output Enable	Active
			0	MCLK		
1	1	0	1	High-Z	Output Enable	Active**
			0	MCLK/M		
X	0	1	1	High-Z	Power-Down	Power-Down
			0	MCLK		Active
X	1	1	1	High-Z	Power-Down	Power-Down
			0	MCLK/M		Active

*This mode is for applications where OUT0 is not used, but CTRL0 is used as a device shutdown.

**Factory default setting.

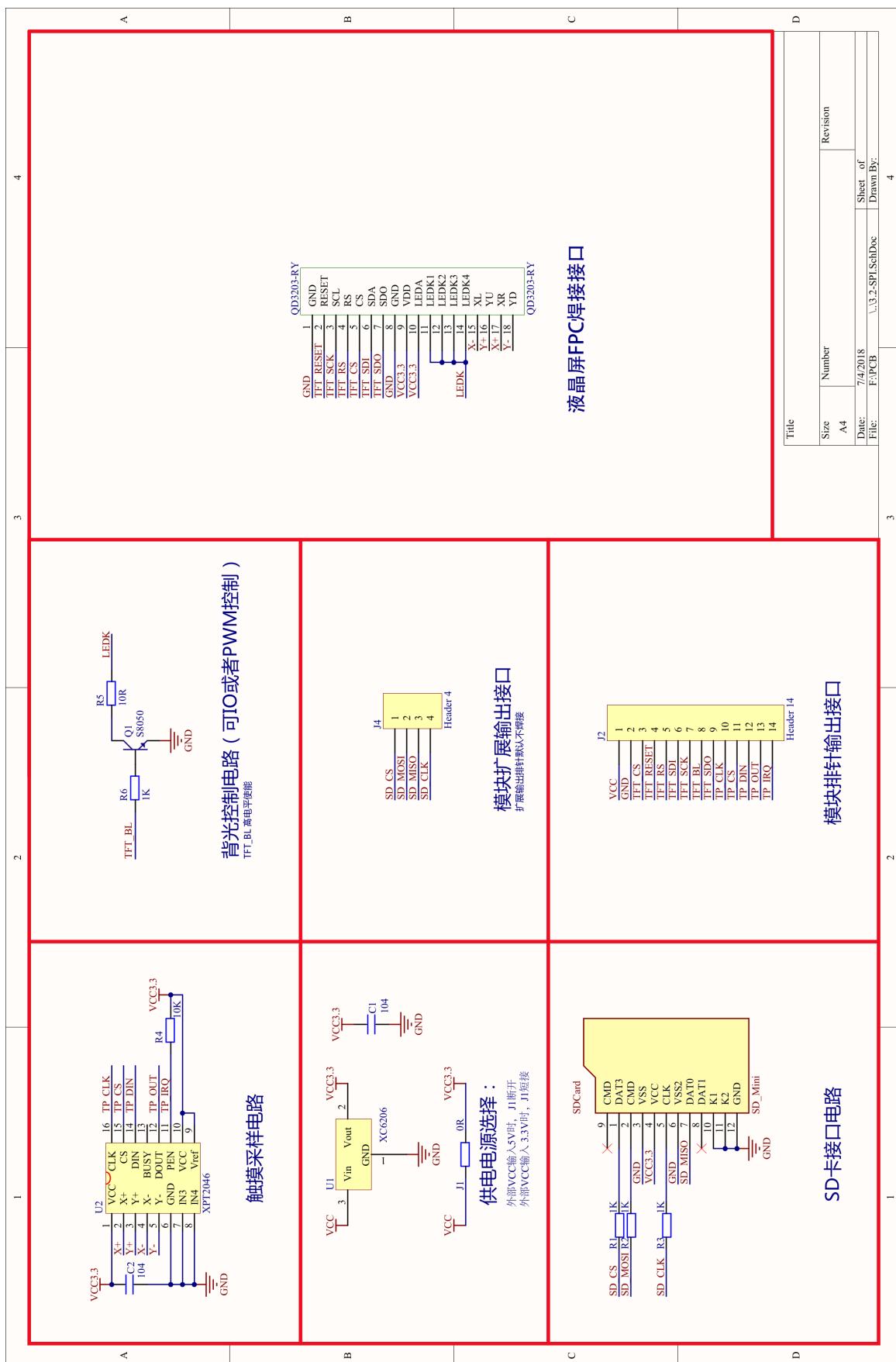
***See standby (power-down) current specification for power-down current range.

фиг. А.3: Таблица 2 от каталожните данни⁽¹³⁾ на DS1085

TABLE 3-1: REGISTER ADDRESSES

Address IOCON.BANK = 1	Address IOCON.BANK = 0	Access to:
00h	00h	IODIRA
10h	01h	IODIRB
01h	02h	IPOLA
11h	03h	IPOLB
02h	04h	GPINTENA
12h	05h	GPINTENB
03h	06h	DEFVALA
13h	07h	DEFVALB
04h	08h	INTCONA
14h	09h	INTCONB
05h	0Ah	IOCON
15h	0Bh	IOCON
06h	0Ch	GPPUA
16h	0Dh	GPPUB
07h	0Eh	INTFA
17h	0Fh	INTFB
08h	10h	INTCAPA
18h	11h	INTCAPB
09h	12h	GPIOA
19h	13h	GPIOB
0Ah	14h	OLATA
1Ah	15h	OLATB

фиг. А.4: Таблица 3-1 от каталожните данни⁽¹⁴⁾ на MCP23017



фиг. А.5: Принципна електрическа схема с периферните елементи на развойната платка на тъчскрийн модула

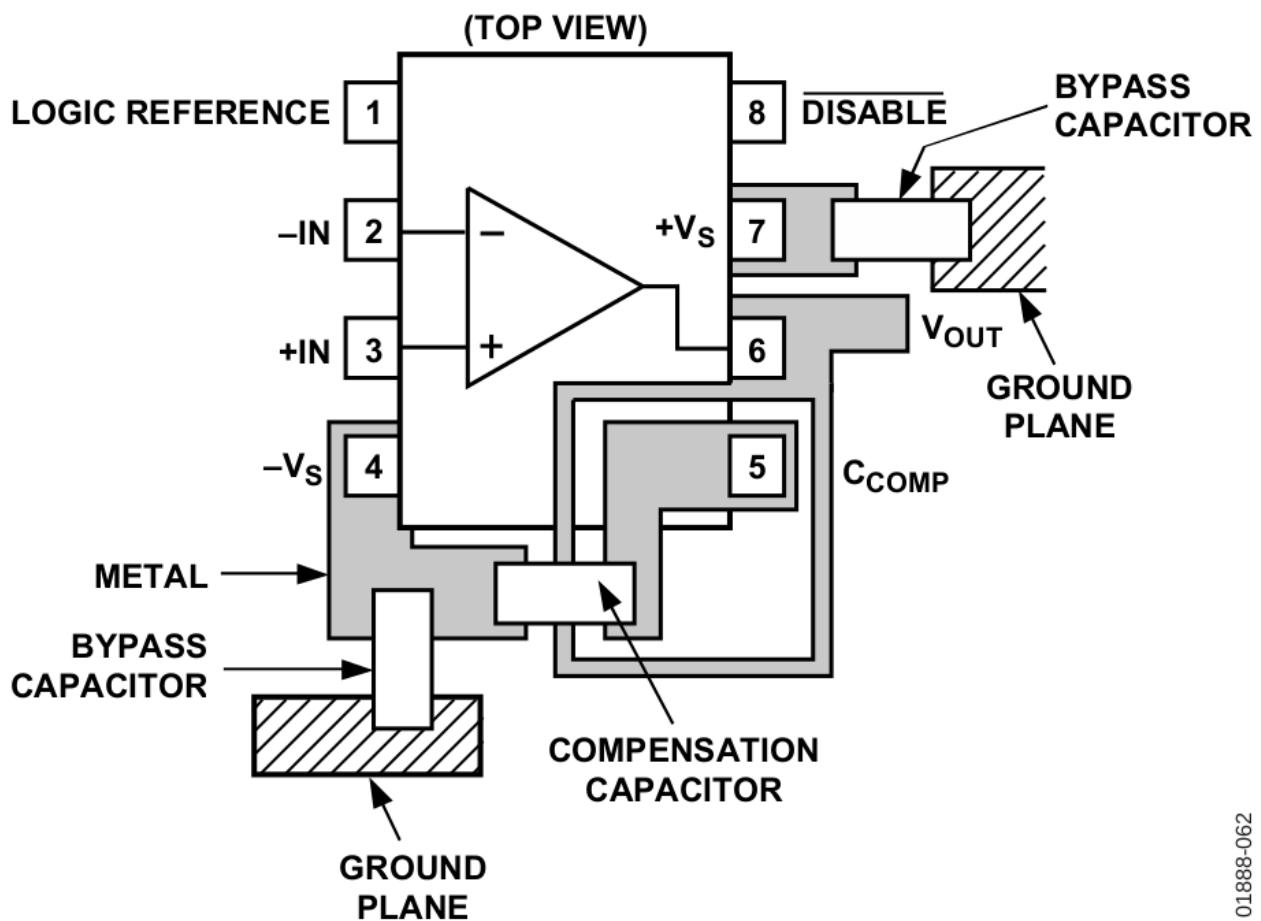
Truth Table

CE	OE	WE	BLE	BHE	I/O₀–I/O₇	I/O₈–I/O₁₅	Mode	Power
H	X	X	X	X	High Z	High Z	Power Down	Standby (I_{SB})
L	L	H	L	L	Data Out	Data Out	Read – All bits	Active (I_{CC})
			L	H	Data Out	High Z	Read – Lower bits only	Active (I_{CC})
			H	L	High Z	Data Out	Read – Upper bits only	Active (I_{CC})
L	X	L	L	L	Data In	Data In	Write – All bits	Active (I_{CC})
			L	H	Data In	High Z	Write – Lower bits only	Active (I_{CC})
			H	L	High Z	Data In	Write – Upper bits only	Active (I_{CC})
L	H	H	X	X	High Z	High Z	Selected, Outputs Disabled	Active (I_{CC})
L	X	X	H	H	High Z	High Z	Selected, Outputs Disabled	Active (I_{CC})

фиг. А.6: Таблица на истинност от каталожните данни⁽⁸⁾ на CY7C1021D**Table 6. Recommended Component Values**

Noise Gain (Noninverting Gain)	R_S (Ω)	R_F (Ω)	R_G (Ω)	C_{COMP} (pF)	Slew Rate (V/μs)	-3 dB SS BW (MHz)	Output Noise (AD8021 Only) (nV/√Hz)	Output Noise (AD8021 with Resistors) (nV/√Hz)
1	75	75	NA	10	120	490	2.1	2.8
2	49.9	499	499	7	150	205	4.3	8.2
5	49.9	1 k	249	2	300	185	10.7	15.5
10	49.9	1 k	110	0	420	150	21.2	27.9
20	49.9	1 k	52.3	0	200	42	42.2	52.7
100	49.9	1 k	10	0	34	6	211.1	264.1

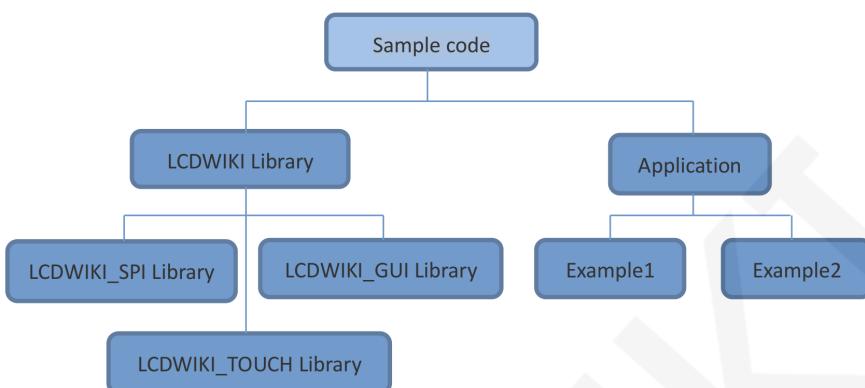
фиг. А.7: Таблица 6 от каталожните данни⁽¹⁵⁾ на AD8021



01888-062

Figure 62. Recommended Location of Critical Components and Guard Ring

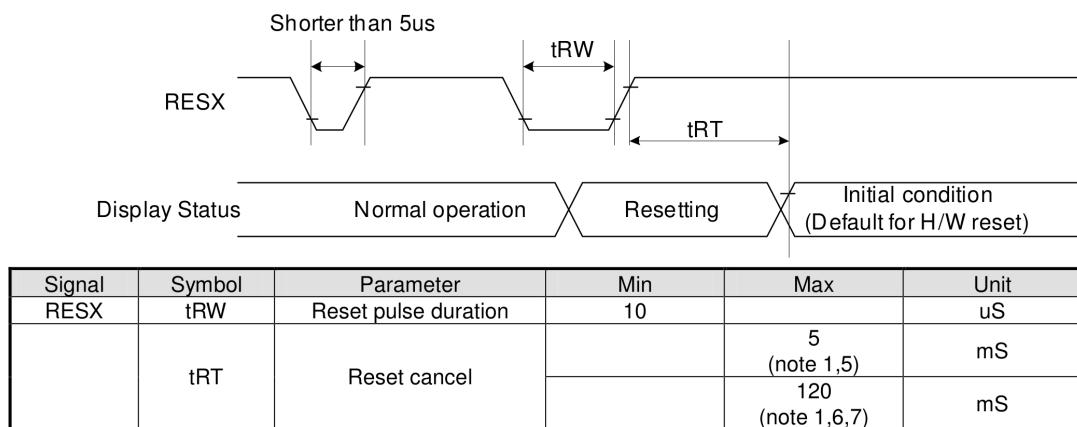
фиг. А.8: Фигура 62 от каталожните данни⁽¹⁵⁾ на AD8021



фиг. А.9: Структура на китайската софтуерна библиотека за тъч скрийн модула

Bit	Name	Description
MY	Row Address Order	
MX	Column Address Order	These 3 bits control MCU to memory write/read direction.
MV	Row / Column Exchange	
ML	Vertical Refresh Order	LCD vertical refresh direction control.
BGR	RGB-BGR Order	Color selector switch control (0=RGB color filter panel, 1=BGR color filter panel)
MH	Horizontal Refresh ORDER	LCD horizontal refreshing direction control.

фиг. A.10: Описание на регистъра **MADCTL** от каталожните данни⁽²⁵⁾ на ILI9341 (виж стр. 127)



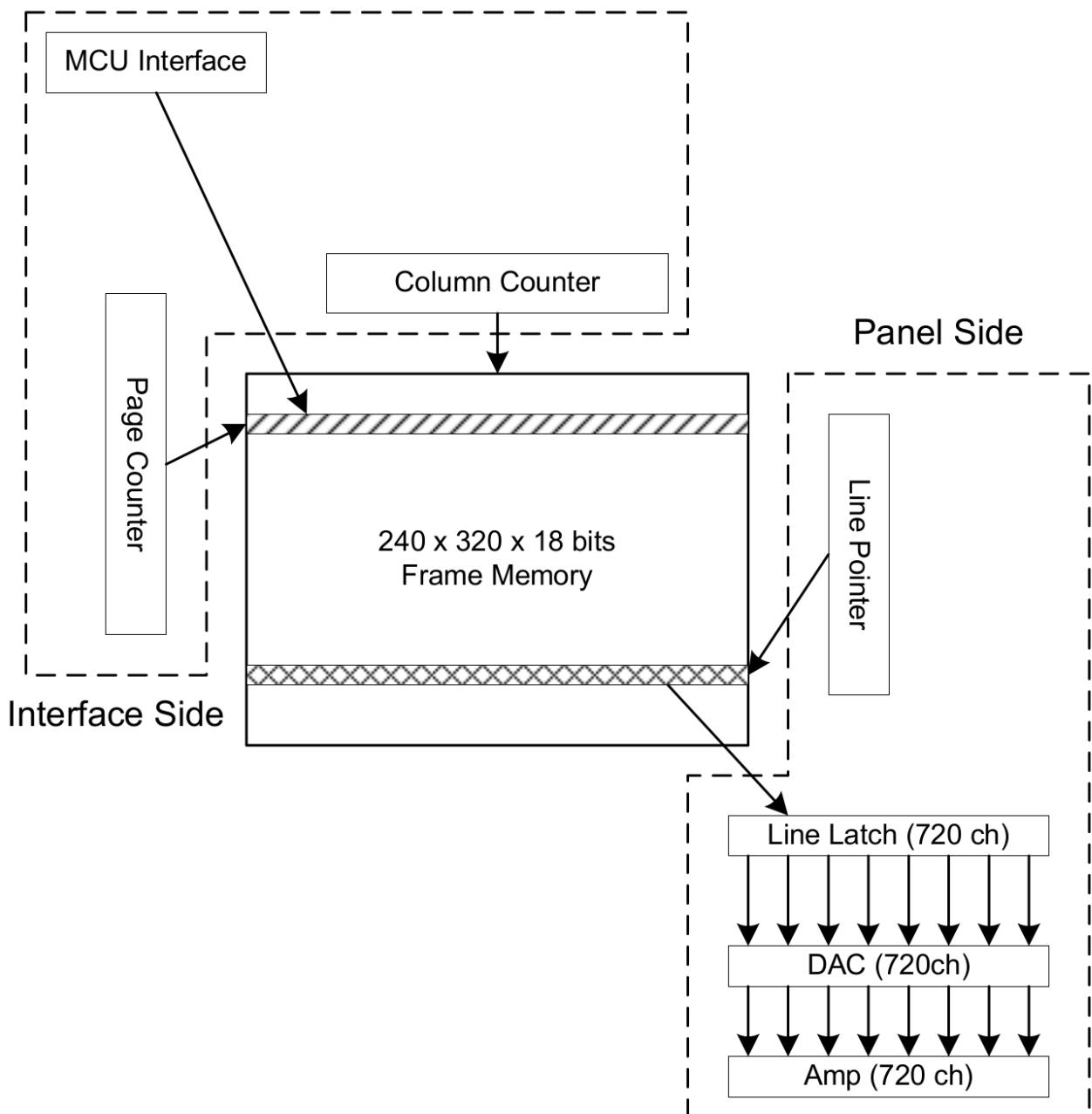
фиг. A.11: Времева диаграма, описваща начинът за презареждане на ILI9341, взета от каталожните му данни⁽²⁵⁾ (стр. 225)

```

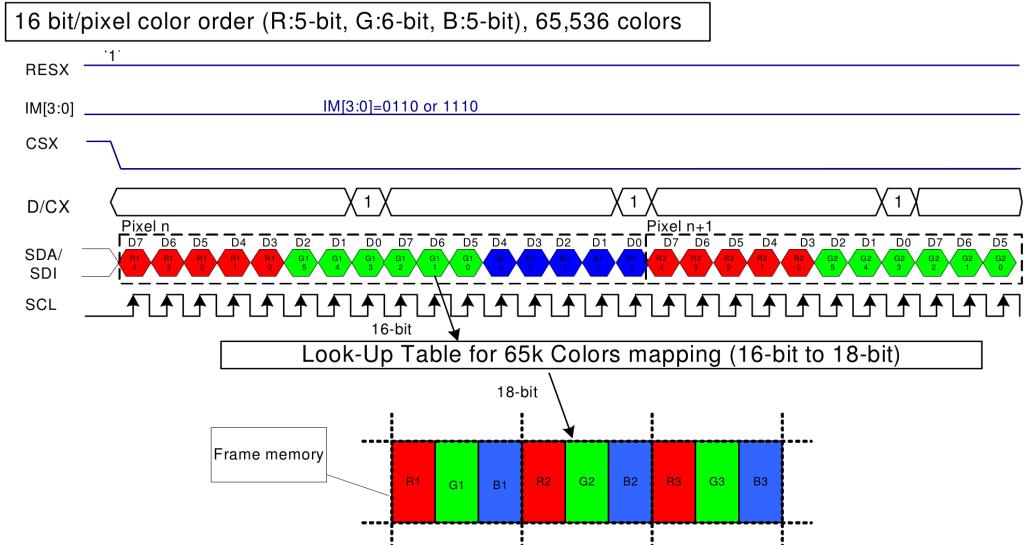
case 0x9341:
    lcd_driver = ID_9341;
    //WIDTH = 240,HEIGHT = 320;
    //width = WIDTH, height = HEIGHT;
    XC=ILI9341_COLADDRSET,YC=ILI9341_PAGEADDRSET,CC=ILI9341_MEMORYWRITE,RC=HX8357_RAMRD,SC1=0x33,SC2=0x3
    static const uint8_t ILI9341_regValues[] PROGMEM =
    {
        // BOE 2.4"
        ILI9341_SOFTRESET,0,                                //Soft Reset
        TFTLCD_DELAY8, 50,
        ILI9341_DISPLAYOFF, 0,                            //Display Off
        // ILI9341_PIXELFORMAT, 1, 0x55,          //Pixel read=565, write=565.
        ILI9341_INTERFACECONTROL, 3, 0x01, 0x01, 0x00, //Interface Control needs EXTC=1 MV_EOR=0,
        ILI9341_POWERCONTROLB, 3, 0x00, 0x81, 0x30, //Power Control B [00 81 30]
        ILI9341_POWERONSEQ, 4, 0x64, 0x03, 0x12, 0x81, //Power On Seq [55 01 23 01]
        ILI9341_DRIVERTIMINGA, 3, 0x85, 0x10, 0x78, //Driver Timing A [04 11 7A]
        ILI9341_POWERCONTROLA, 5, 0x39, 0x2C, 0x00, 0x34, 0x02, //Power Control A [39 2C 00 34
        ILI9341_RUMPRATIO, 1, 0x20,           //Pump Ratio [10]
        ILI9341_DRIVERTIMINGB, 2, 0x00, 0x00, //Driver Timing B [66 00]
        ILI9341_RGBSIGNAL, 1, 0x00,          //RGB Signal [00]
        // ILI9341_FRAMECONTROL, 2, 0x00, 0x1B,      //Frame Control [00 1B]
        //          0xB6, 2, 0xA, 0xA2, 0x27, //Display Function [0A 82 27 XX] .kbv SS=1
        ILI9341_INVERSIONCONRTOL, 1, 0x00,          //Inversion Control [02] .kbv NLA=1, NLB=1, NLC=1
        ILI9341_POWERCONTROL1, 1, 0x21,           //Power Control 1 [26]
        ILI9341_POWERCONTROL2, 1, 0x11,           //Power Control 2 [00]
        ILI9341_VCOMCONTROL1, 2, 0x3F, 0x3C, //VCOM 1 [31 3C]
        ILI9341_VCOMCONTROL2, 1, 0xB5,           //VCOM 2 [C0]
        ILI9341_MEMCONTROL, 1, ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR,
        ILI9341_PIXELFORMAT, 1, 0x55,          //Pixel read=565, write=565.
        ILI9341_FRAMECONTROL, 2, 0x00, 0x1B,      //Frame Control [00 1B]
        ILI9341_MEMORYACCESS, 1, 0x48,          //Memory Access [00]
        ILI9341_ENABLE3G, 1, 0x00,           //Enable 3G [02]
        ILI9341_GAMMASET, 1, 0x01,           //Gamma Set [01]
        ILI9341_UNDEFINE0, 15, 0x0f, 0x26, 0x24, 0x0b, 0x0e, 0x09, 0x54, 0xa8, 0x46, 0x0c, 0x17, 0x0
        ILI9341_UNDEFINE1, 15, 0x00, 0x19, 0x1b, 0x04, 0x10, 0x07, 0x2a, 0x47, 0x39, 0x03, 0x06, 0x0
        ILI9341_ENTRYMODE, 1, 0x07,          //Entry Mode
        ILI9341_SLEEPOUT, 0,                //Sleep Out
        TFTLCD_DELAY8, 150,
        ILI9341_DISPLAYON, 0,              //Display On
    };
    init_table8(ILI9341_regValues, sizeof(ILI9341_regValues));
    break;
}

```

фиг. А.12: Откъс, показващ поредицата от команди за настройване на драйверния чип ILI9341 след захранване на тъчскрийн модула, от библиотеката „LCDWIKI_SPI“ на LCDWIKI⁽³⁸⁾

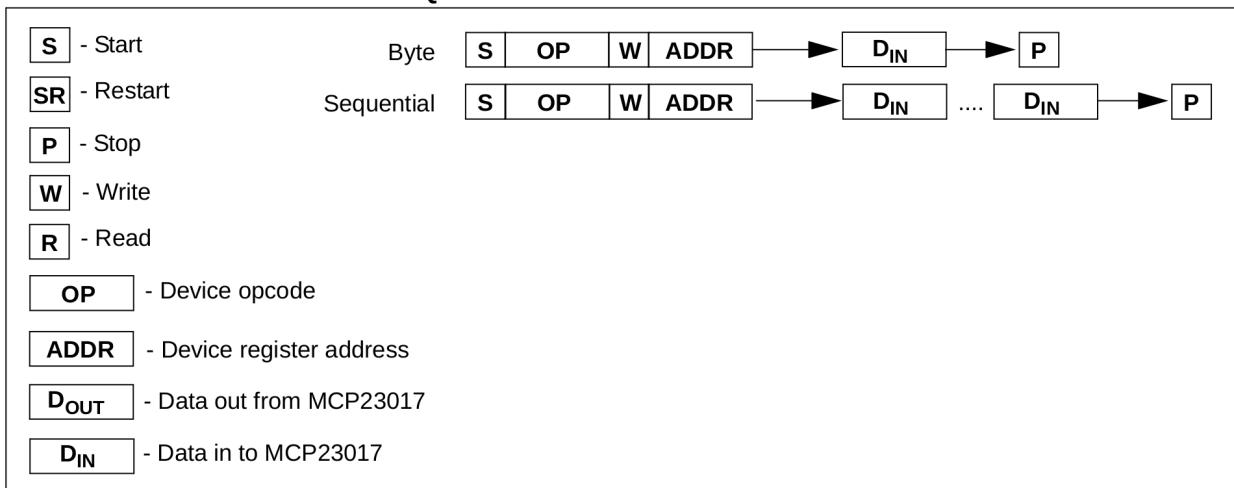


фиг. А.13: Структура на вградената графична памет (GRAM) на ILI9341, взета от каталожните данни⁽²⁵⁾ (стр. 202) на интегралната схема



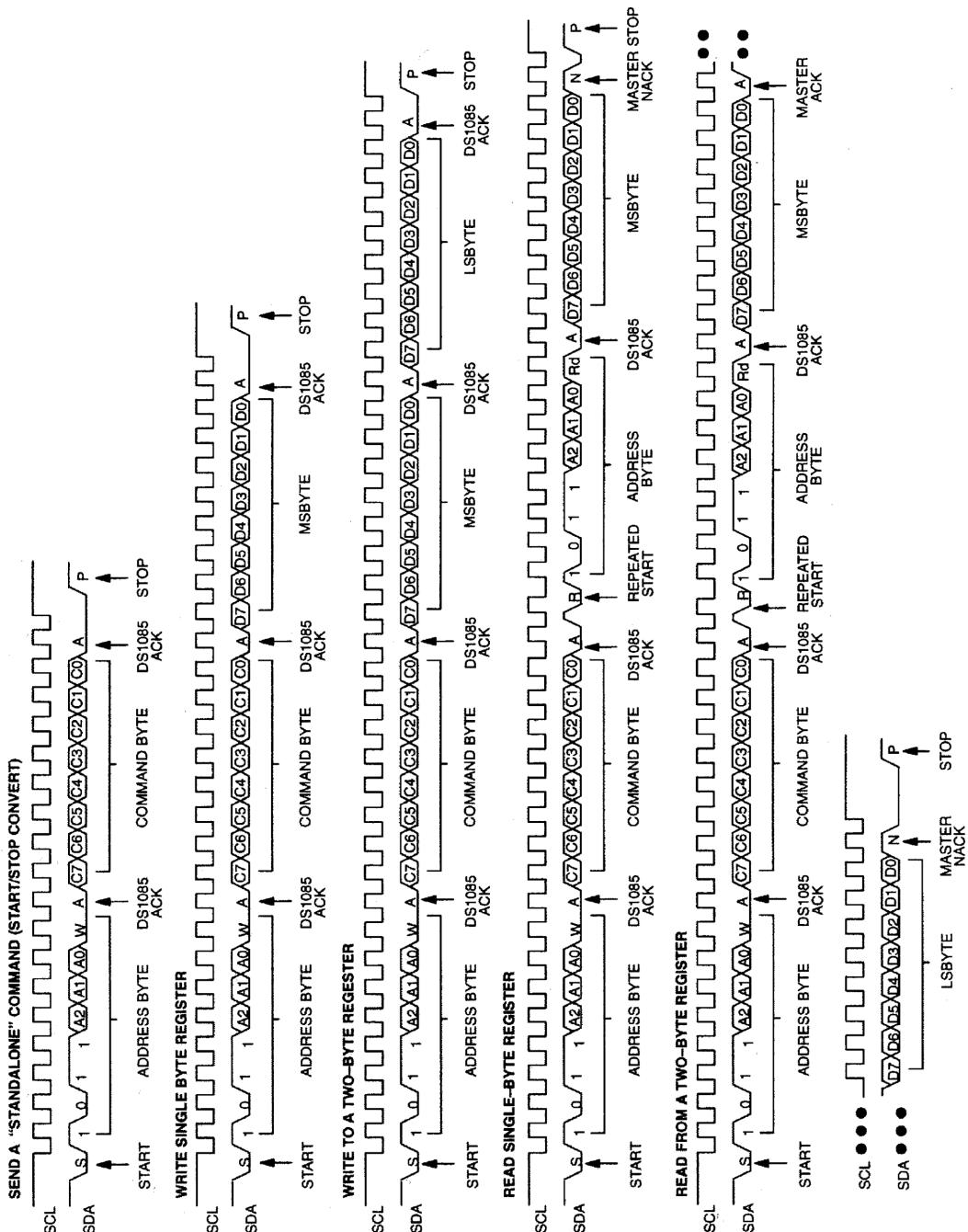
фиг. А.14: Времедиаграма от каталожните данни⁽²⁵⁾ на ILI9341, показваща начина на обработка на 16-битов цвят, когато интегралната схема е настроена в режим 16 бита/пиксел

FIGURE 3-1: BYTE AND SEQUENTIAL WRITE



фиг. А.15: Фигура 3-1 от каталожните данни⁽¹⁴⁾ на MCP23017

Figure 4. 2-WIRE SERIAL COMMUNICATION WITH DS1085



фиг. А.16: Фигура 4 от каталожните данни⁽¹³⁾ на DS1085

Table 8. PROGRAMMABLE DIVISOR N VALUES

BIT VALUE	DIVISOR (N)
00000000 00XXXXXX	2*
00000000 01XXXXXX	3
—	—
—	—
—	—
—	—
11111111 11XXXXXX	1025

*Factory Default Setting

фиг. А.17: Таблица 8 от каталожните данни⁽¹³⁾ на DS1085

```

1 //for resolution 240x320, the calibration parameter is 663, -13, 894, -30
2 //for resolution 320x480, the calibration parameter is 852, -14, 1284, -30
3
4 #define XFAC      663 //852
5 #define XOFFSET   (-13) //(-14)
6 #define YFAC      894 //1284
7 #define YOFFSET   (-30)
...
10 else if(TP_Read_Coordinate2(&x,&y)) //screen coordinate
11 {
12 /*
13     long temp;
14     temp = (long)XFAC*x/10000;
15     x=temp+XOFFSET;
16     temp = (long)YFAC*y/10000;
17     y=temp+YOFFSET;
18 */
19     x=((long)XFAC*x)/10000+XOFFSET;
20     y=((long)YFAC*y)/10000+YOFFSET;
...

```

фиг. А.18: Откъси от програмния код на библиотеката „LCDWIKI_TOUCH“, показваща преобразуването на данни от XPT2046 в координати на LCD экрана

Table 30-15. Pin Mapping Serial Programming

Symbol	Pins (TQFP-100)	Pins (TQFP-64)	I/O	Description
PDI	PB2	PE0	I	Serial Data in
PDO	PB3	PE1	O	Serial Data out
SCK	PB1	PB1	I	Serial Clock

фиг. А.19: Таблица 30-15 от каталожните данни⁽⁷⁾ на ATmega2561