Keep learning.
It's the Washington Way.

EMBSYS 105
Programming with Embedded & Real-Time Operating Systems

Instructor: Nick Strathy, nstrathy@uw.edu

TA: Gideon Lee, gideonhlee@yahoo.com

Lecture 7 2/24/2020

PROFESSIONAL & CONTINUING EDUCATION

UNIVERSITY of WASHINGTON

Looking ahead

Date	Lecture number	Assignment
1/6	L1	A1 due* before L2
1/13	L2	A2 due before L3
1/20	L3	A3 due before L4
1/27	L4	A4 due before L5
2/3	L5	A5 due before L7, Project due before L10
2/10	L6	
2/17	Holiday – enjoy!	
2/24	L7	
3/2	L8, Guest Lecture: Bill Lamie (CEO) & Yuxin Zhou VP Eng of Express Logic	
3/9	L9	
3/16	L10 – Student presentations	

* Assignments are due Sunday night at 11:59 PM

Previous Lecture (L6) Overview

- Connecting jumper wires to make the LCD work
- Event driven MP3 player
- Task priorities - Rate monotonic scheduling
- Adding I2C driver to PJDF
- Tracing uCOS message queue
- Explore graphics library – Button class

Current Lecture (L7) Overview

- Assignment 5 Touch Driver
- Project design suggestions
- GenerateMp3Header.ps1 tool
- Audacity tool
- Reducing Code Memory footprint
- Communicating with the vs1053 Chip
- Fun: previous generation MP3 Player
- SD/MMC
- FAT File System
- Read a directory with Arduino SD library

LCD/wires issue

- Did everyone receive their wires? If not let me know ASAP.
- With the wires (and extra headers), your LCD should now work.

Assignment 5 Touch Driver

- At this point in the course, you should have a touch driver working.
- If your PJDF driver is not working yet
  - You should put it on hold and come back to it later if there’s time
  - Move on to the project using your non-PJDF driver

Assignment 5 Touch Driver

Touch should work when starting up the application in the debugger (assuming you have I2C initialized and you disable interrupts for I2C transactions)

Summary of hardware issues with touch driver:

- Due to hardware changes in this year's board, touch hangs if
  - you start up the board without the debugger
  - you use multi-touch
  - **You are not required to fix these issues**

Assignment 5 Touch Driver

However, to make touch work when not starting up in the debugger:

- Remove the jumper underneath the upper right corner of your Nucleo and use it to connect C1 to R4 beside the USB connector on the Nucleo.
- Add a delay like this in main():

```
Hw_init();
delay(100000);
I2C1_init();
```

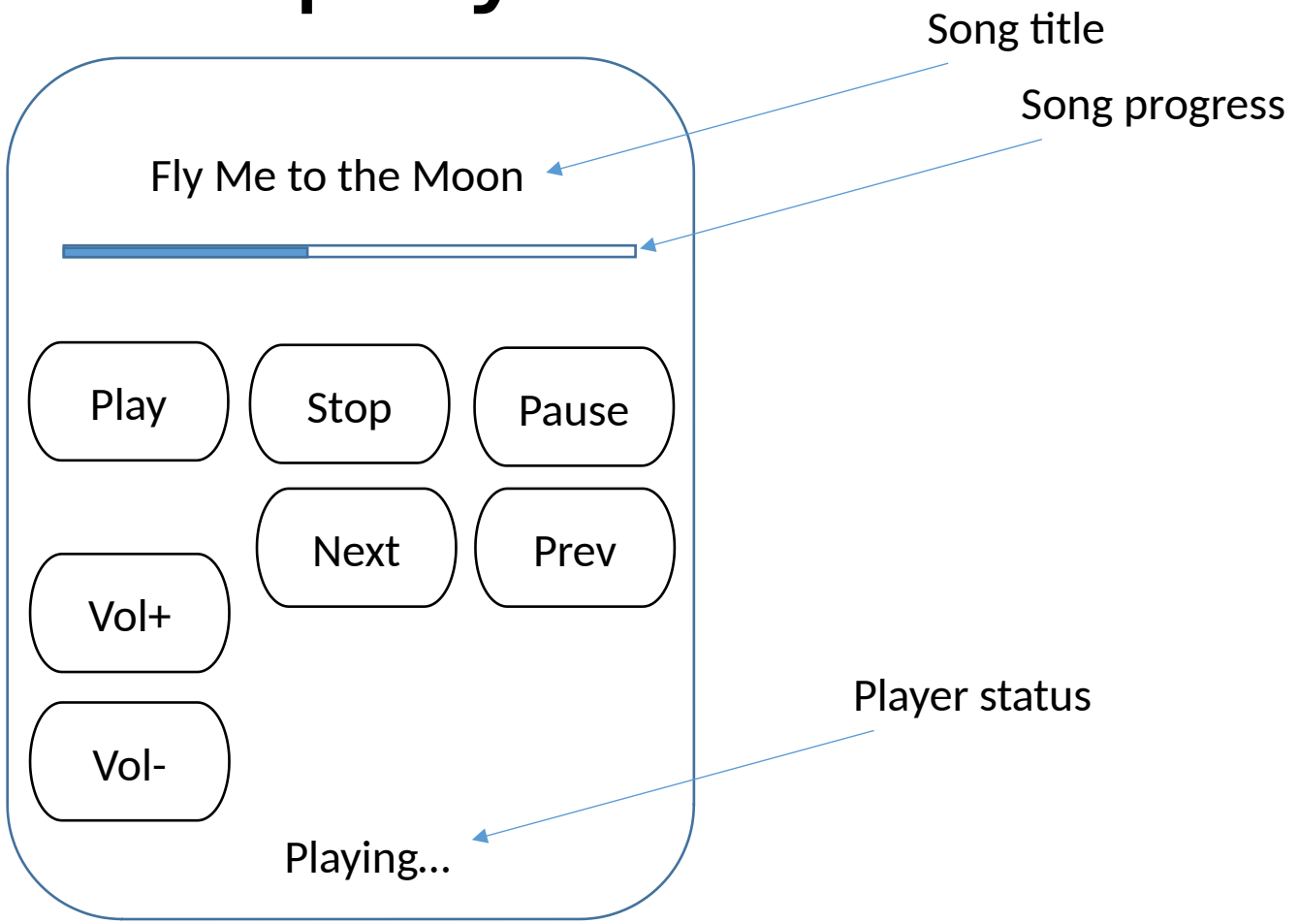
Assignment 5 Touch Driver

Sketch for a workaround for multi-touch hang:

- Add code to time-out when checking the I2C status
- If timeout happens reset I2C hardware

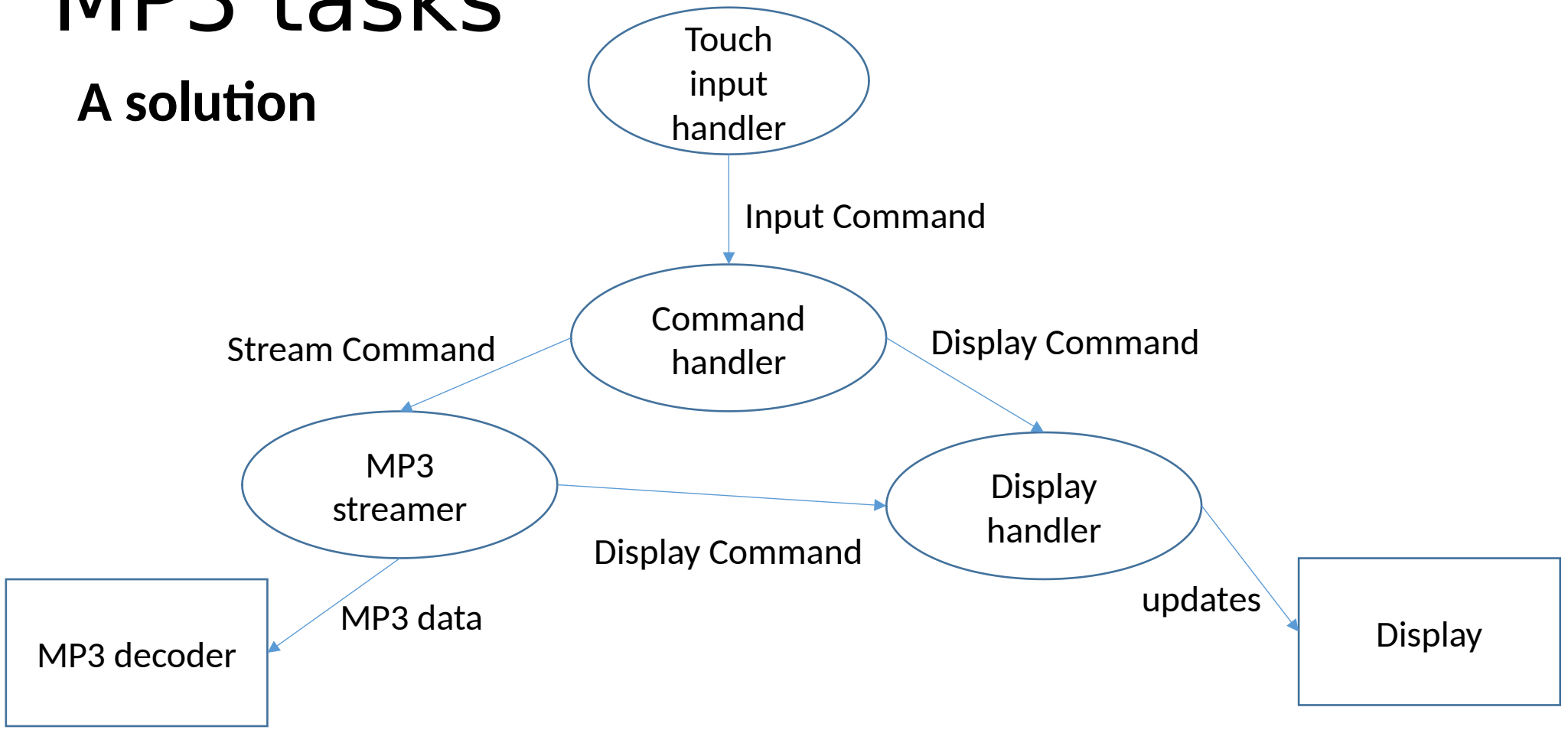
MP3 Player Display

A solution



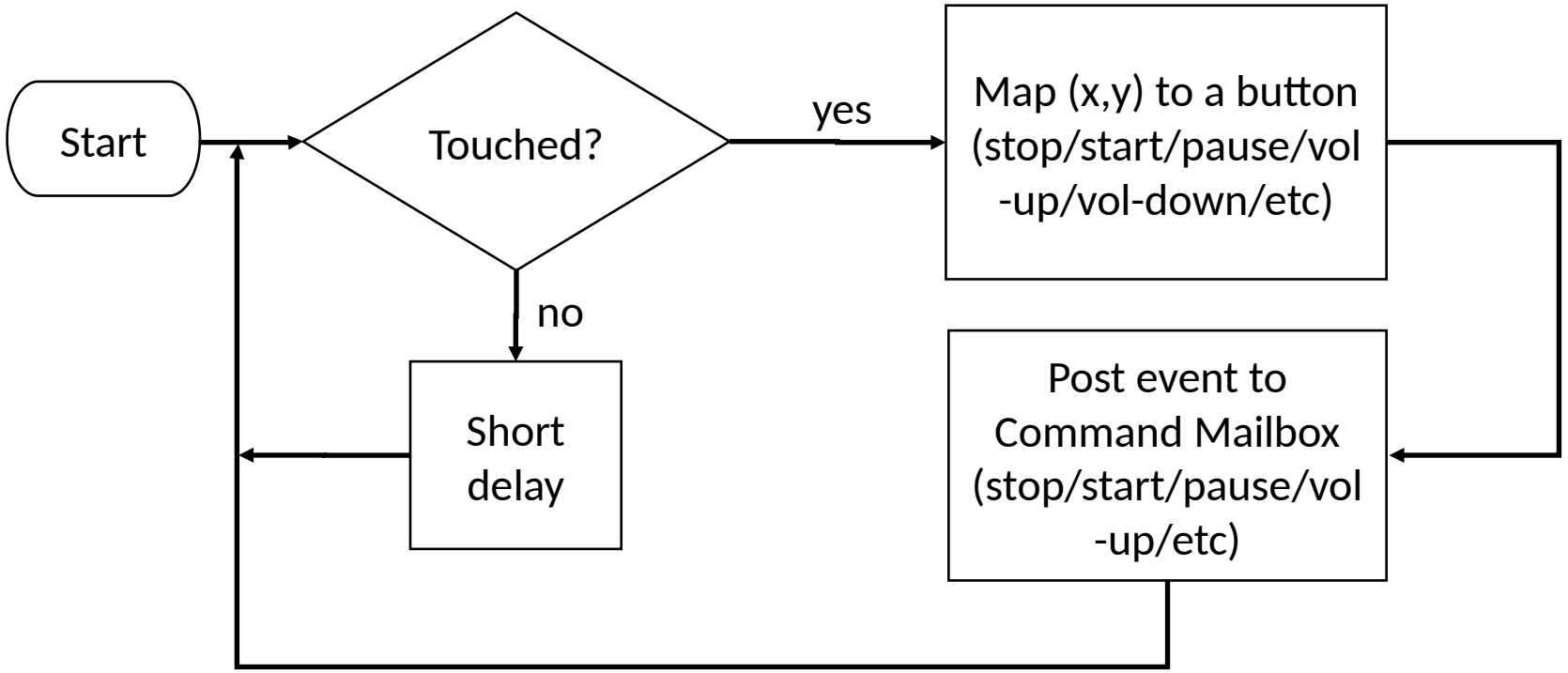
MP3 tasks

A solution



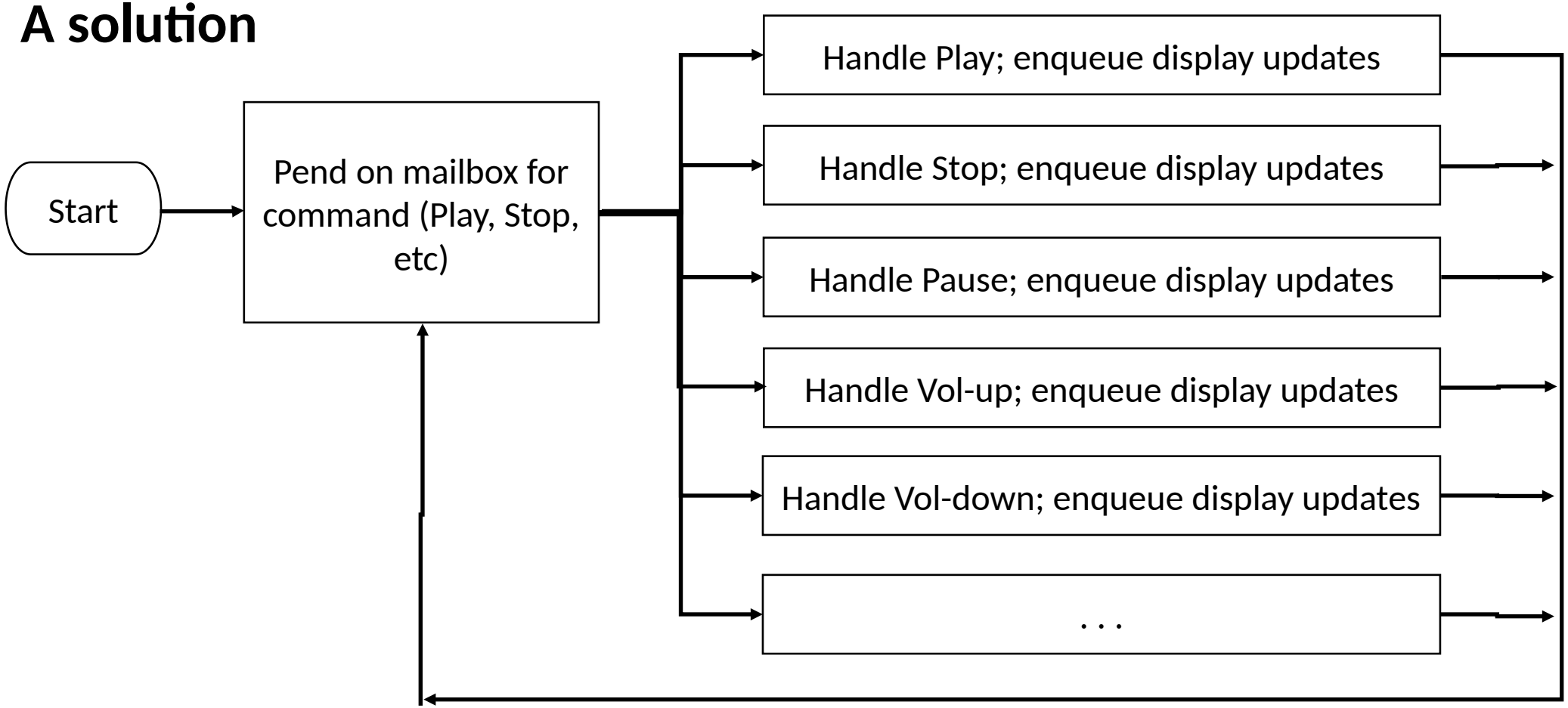
Touch input task

A solution



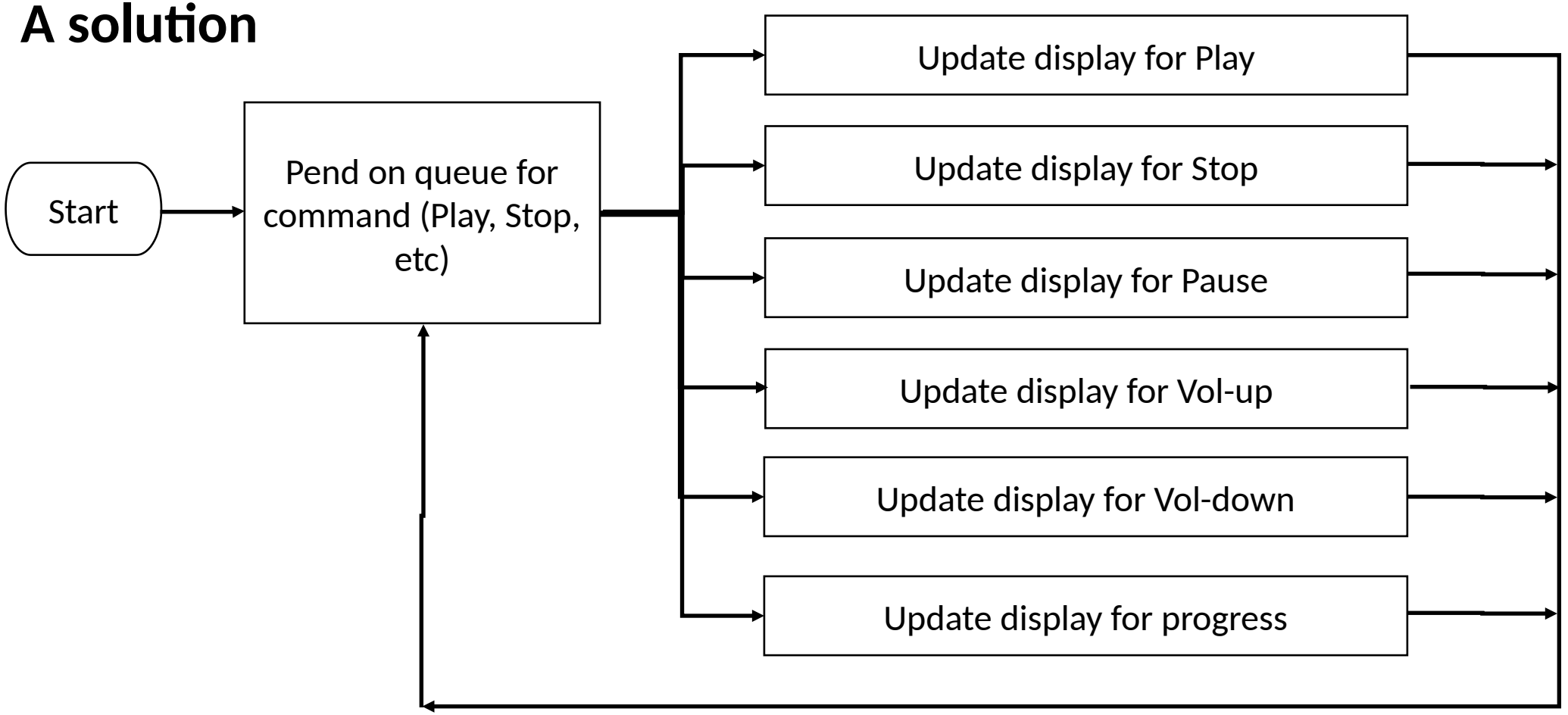
Command handler task

A solution



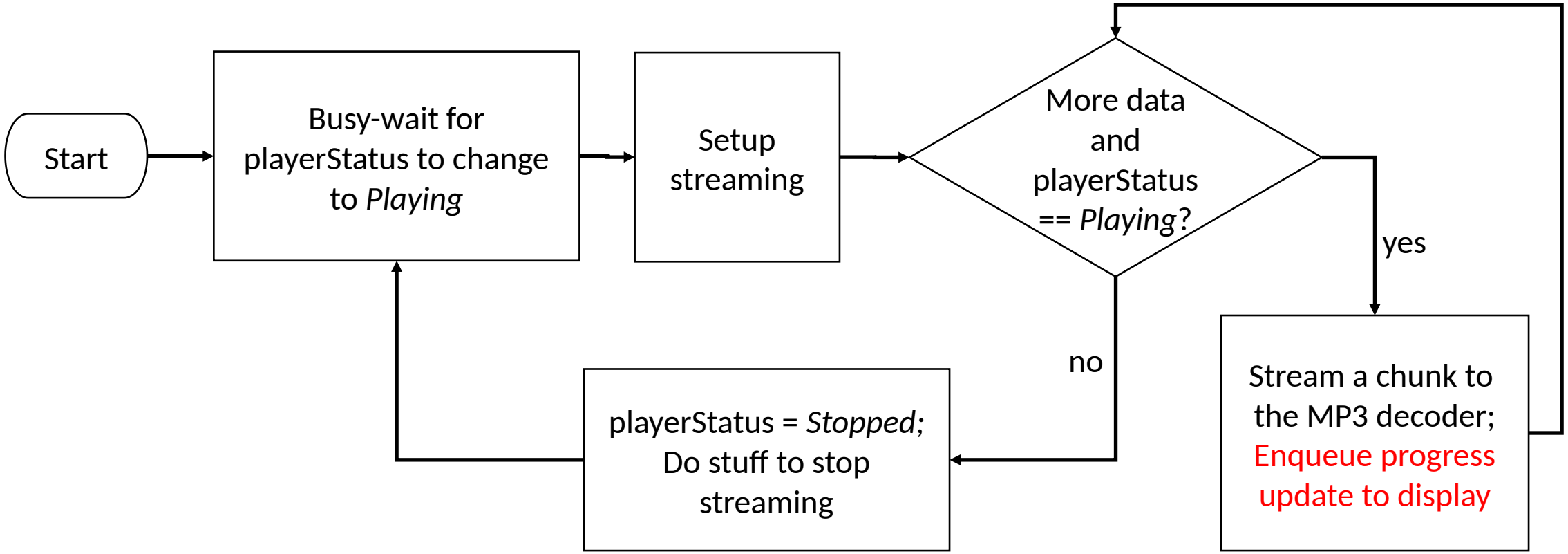
Display handler task

A solution



Streaming task

Play/Stop flow chart with progress update – a solution



Demo GenerateMp3Header.ps1

Powershell script in MP3Player starter app:

- Script location in the project: MP3data/GenerateMp3Header.ps1
- You may need to allow PS scripts to run on your machine, so launch Powershell in administrator mode:
  - Locate Powershell app in start menu
  - Right-click/Run as Administrator
  - To allow Powershell scripts to run enter this command:
    - Set-ExecutionPolicy Bypass
- Powershell has an inconvenient way of determining the current path, so just use full filepathnames as shown in the example in the script

Audacity

Audacity® is free, open source, cross-platform audio **software** for multi-track recording and editing.

- Can select a portion of audio track and “Export audio …”
- Can select bit rate for MP3 export

MP3 bit rates

Handling high MP3 bit rates

- MP3 is a lossy audio compression format
- Bit rate can vary from file to file, even within a file
- Highest standard bit rate is 320 kbit/s
- Problem: we have up to 3 devices contending for exclusive access to the SPI bus (MP3 chip, LCD, SD)

MP3 bit rates

Handling high MP3 bit rates

- Vs1053 chip asserts DREQ high when it is ready for data
- **It will accept a minimum of 32 bytes when DREQ is high**
- Problem: if the file is encoded at 320 kbit/s it will take as little as 0.8 ms to play 32 bytes – result: 0.8 ms music then we need another 32 bytes
- However, Vs1053 internal buffer holds up to 2048 bytes
- Solution: we keep feeding 32-bytes to the decoder while DREQ is high and only delay the streaming task when DREQ goes low.

MP3 bit rates

Logic to allow high MP3 bit rates

- Low level streaming logic (built in to PJDF driver):
  - Repeat:
    - while DREQ is low
      - delay *some* ticks (allow lower priority tasks to run)
    - send 32 bytes

Reducing code footprint

- We are limited to 32kB *Code* by the IAR free license
- Find your project's memory usage in .map file in the EWARM project Output folder
- Can remove uCOS unused features in App/uCOS/os_cfg.h
  - Maybe reclaim 100's of bytes
- SD card driver takes ~7kB of code if you use it, you may have to eliminate some other features – trade-off.
- If IAR Tiny stdio printf were not so small we could use our built-in printf
  - Caveat: ours is not thread safe, max int is 16 bits

Communicating with the vs1053 Chip

Communicating with the vs1053 Chip

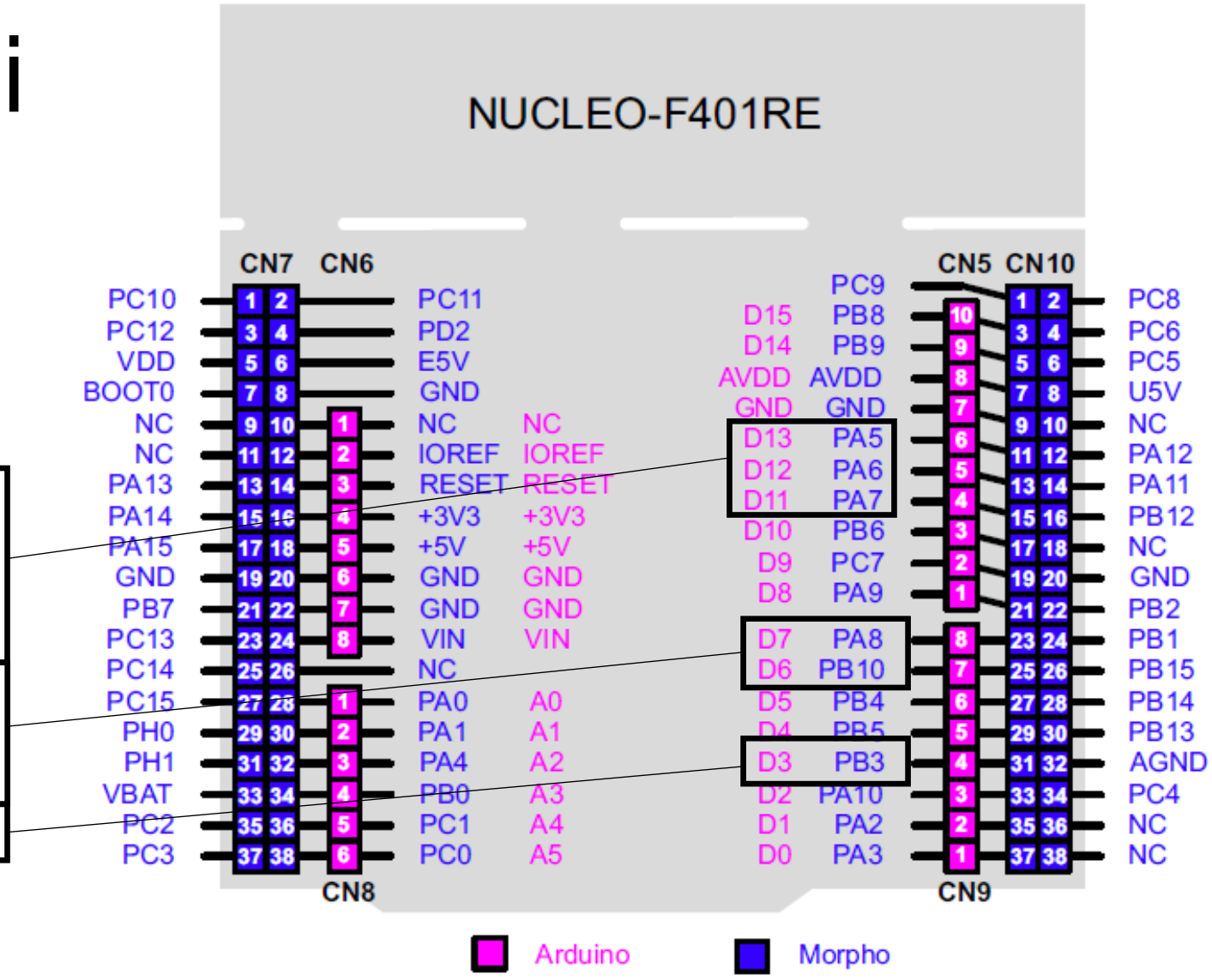
- References
  - **Adafruit Music Maker Shield**: https://learn.adafruit.com/downloads/pdf/adafruit-music-maker-shield-vs1053-mp3-wav-wave-ogg-vorbis-player.pdf
  - **User manual STM32 Nucleo-64 boards**: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00105823.pdf
  - **VS1053b Datasheet**: http://www.vlsi.fi/fileadmin/datasheets/vs1053.pdf
- The application controls the vs1053 chip by sending commands to the chip's **Serial Command Interface (SCI)** bus via SPI1 bus connected to the host.
- MP3 data is sent to the chip's **Serial Data Interface (SDI)** bus also via SPI1.
- We control which bus is selected by the GPIO chip select lines defined in bspMp3.h:
  - MP3_VS1053_MCS_GPIO/_Pin : assert this low to send commands to the chip
  - MP3_VS1053_DCS_GPIO/_Pin: assert this low to send data to the chip
- A 3rd GPIO is used by the vs1053 to notify the host when more data can be sent to the decoder:
  - MP3_VS1053_DREQ_GPIO/_Pin: asserted high by vs1053 when ready to receive data

Communicating with the vs1053 Chip

- There are three 'totally fixed' pins, the hardware SPI pins:
  - **SPI SCK** - connected to Digital #13 (but can be connected to the ISP header with a jumper) used by both the SD card and VS1053
  - **SPI MISO** - connected to Digital #12 (but can be connected to the ISP header with a jumper) used by both the SD card and VS1053
  - **SPI MOSI** - connected to Digital #11 (but can be connected to the ISP header with a jumper) used by both the SD card and VS1053
- There are a couple of other pins that are required for talking to the VS1053 to play MP3s and such
  - **MCS** - this is the VS1053 chip select pin, connected to Digital #7
  - **DCS** - this is the VS1053 data select pin, connected to Digital #6
  - **CCS** - this is the SD Card chip select pin, connected to Digital #4
  - **DREQ** - this is the VS1053 data request interrupt pin - connected to digital #3

Communicati with the vs1053 Chip

SPI SCK – D13 PA5 SPI MISO – D12 PA6 SPI MOSI – D11 PA7
MCS – D7 PA8 DCS – D6 PB10
DREQ – D3 PB3



Communicating with the vs1053 Chip

Put chip in command mode

```
Ioctl(hMp3, PJDF_CTRL_MP3_SELECT_COMMAND, 0, 0);
```

Meaning: Our MP3 driver will ensure that MCS is asserted for any subsequent writes to the vs1053

Put chip in data mode

```
Ioctl(hMp3, PJDF_CTRL_MP3_SELECT_DATA, 0, 0);
```

Meaning: Our MP3 driver will ensure that DCS is asserted for any subsequent writes to the vs1053

Communicating with the vs1053 Chip

Soft chip reset

Reset the chip by sending the serial command bytes in const char array BspMp3SoftReset[]:

```
const INT8U BspMp3SoftReset[] = { 0x02, 0x00, 0x08, 0x04 };
```

0x02 is the SCI *write register* command (see p. 21 Data Sheet)

0x00 is the address of the SCI_MODE register (see p. 38 Data Sheet)

0x08, 0x04 specifies the 16-bit register value with most significant byte first. Two bits are set to: cancel decoding any current file (SM_CANCEL), and reset for new serial data SM_SDINEW (see p. 39 Data Sheet).

Communicating with the vs1053 Chip

Serial Protocol for Serial Command Interface (SPI/SCI)
p. 21 of VS1053b Datasheet

Instruction		
Name	Opcode	Operation
READ	0b0000 0011	Read data
WRITE	0b0000 0010	Write data

Communicating with the vs1053 Chip

p. 38 of VS1053b Datasheet

SCI registers, prefix SCI_					
Reg	Type	Reset	Time[1]	Abbrev[bits]	Description
0x0	rw	0x4000[6]	80 CLKI[4]	MODE	Mode control
0x1	rw	0x000C[3]	80 CLKI	STATUS	Status of VS1053b
0x2	rw	0	80 CLKI	BASS	Built-in bass/treble control
0x3	rw	0	1200 XTALI[5]	CLOCKF	Clock freq + multiplier
0x4	rw	0	100 CLKI	DECODE_TIME	Decode time in seconds
0x5	rw	0	450 CLKI[2]	AUDATA	Misc. audio data
0x6	rw	0	100 CLKI	WRAM	RAM write/read
0x7	rw	0	100 CLKI	WRAMADDR	Base address for RAM write/read
0x8	r	0	80 CLKI	HDAT0	Stream header data 0
0x9	r	0	80 CLKI	HDAT1	Stream header data 1
0xA	rw	0	210 CLKI[2]	AIADDR	Start address of application
0xB	rw	0	80 CLKI	VOL	Volume control
0xC	rw	0	80 CLKI[2]	AICTRL0	Application control register 0
0xD	rw	0	80 CLKI[2]	AICTRL1	Application control register 1
0xE	rw	0	80 CLKI[2]	AICTRL2	Application control register 2
0xF	rw	0	80 CLKI[2]	AICTRL3	Application control register 3

Communicating with the vs1053 Chip

p. 39 of VS1053b Datasheet

Bit	Name	Function	Value	Description
0	SM_DIFF	Differential	0 1	normal in-phase audio left channel inverted
1	SM_LAYER12	Allow MPEG layers I & II	0 1	no yes
2	SM_RESET	Soft reset	0 1	no reset reset
3	SM_CANCEL	Cancel decoding current file	0 1	no yes
4	SM_EARSPEAKER_LO	EarSpeaker low setting	0 1	off active
5	SM_TESTS	Allow SDI tests	0 1	not allowed allowed
6	SM_STREAM	Stream mode	0 1	no yes
7	SM_EARSPEAKER_HI	EarSpeaker high setting	0 1	off active
8	SM_DACT	DCLK active edge	0 1	rising falling
9	SM_SDIORD	SDI bit order	0 1	MSb first MSb last
10	SM_SDISHARE	Share SPI chip select	0 1	no yes
11	SM_SDINEW	VS1002 native SPI modes	0 1	no yes
12	SM_ADPCM	PCM/ADPCM recording active	0 1	no yes
13	-	-	0 1	right wrong
14	SM_LINE1	MIC / LINE1 selector	0 1	MICP LINE1
15	SM_CLK_RANGE	Input clock range	0 1	12..13 MHz 24..26 MHz

Communicating with the vs1053 Chip

Play an MP3 file

Configure MP3 driver to send a command on the serial command interface bus:

```
Ioctl(hMp3, PJDF_CTRL_MP3_SELECT_COMMAND, 0, 0);
```

Send serial command interface command to enter play mode:

```
const INT8U BspMp3PlayMode[] =  { 0x02, 0x00, 0x08, 0x00 };
```

0x02 is the SCI *write register* command (see p. 21 Data Sheet)
0x00 is the address of the SCI_MODE register (see p. 38 Data Sheet)
0x08, 0x00 specifies the 16-bit register value with most significant byte first. One bit is set to reset for new serial data SM_SDINEW (see p. 39 Data Sheet).

Communicating with the vs1053 Chip

Play an MP3 file cont'd

Select serial data interface bus by configuring as follows:

```
Ioctl(hMp3, PJDF_CTRL_MP3_SELECT_DATA, 0, 0);
```

Now we can send bytes of MP3 data to the vs1053 chip by writing to the MP3 driver and they will get sent to the serial data interface on the vs1053 chip and converted to audio signals by the chip.

DREQ – we wait for this signal to assert high before sending the next chunk of 32 bytes to the decoder.

Demo of previous generation MP3 Player

SD/MMC

SD/MMC

Introduction

- SD is a popular flash memory technology that evolved from MMC
- The SD Standard is an “open” standard maintained by the **Secure Digital Association (**www.sdcard.org**)** – open, yes, but have to purchase licence
- MMC – **M**ulti**M**edia **C**ard technology developed by SanDisk and Siemens in 1997.
- SD – **S**ecure **D**igitial card extends MMC technology – developed by SanDisk, Matsushita (Panasonic), and Toshiba, and introduced in 1999
- MMC cards are harder to find now than SD cards. We will focus mainly on SD.

SD/MMC

- SD/MMC Physical comparison:
  - SD slots will accept MMC cards.
  - SD has 9 pins, MMC has 7
  - SD usually has a physical sliding write-protect lock, MMC does not
- Predecessor technology worth mentioning: CompactFlash first introduced by SandDisk in 1994 predates MMC and SD. Still used for high end video. Larger form factor than SD/MMC.
- SD cards come in 3 form factors: standard, mini, micro
- Capacities:
  - SD – standard capacity: 1 – 2 GB, sometimes 4 GB
  - SDHC (2006) – high capacity: up to 32 GB
  - SDXC (2009) – extended capacity: up to 2 TB

- Clockwise from top left:
- 32 MB CompactFlash card: a related technology worth mentioning. CompactFlash first introduced by SandDisk in 1994 predates MMC and SD. Still used for high end video.
- standard form factor 2 GB SD card
- 4 GB micro SDHC card
- 8 GB SDHC card
- 25-cent coin for scale

SD – Speed

Class	Minimum performance	Application
Class 2	2 MB/s	SD video recording
Class 4	4 MB/s	High-definition video (HD) recording including Full HD (from 720p to 1080p/1080i)
Class 6	6 MB/s	
Class 10	10 MB/s	Full HD (1080p) video recording and consecutive recording of HD stills (high-speed data bus)
UHS Speed Class 1 (U1)	10 MB/s	Real-time broadcasts and large HD video files (UHS bus)
UHS Speed Class 3 (U3)	30 MB/s	4K video files (UHS bus)

Source: Wikipedia

SD

- SDIO – another family of SD devices
  - not simply a memory card but a general purpose I/O scheme using the SD interface
  - for devices like wireless adapters, Bluetooth, barcode scanners, etc.
  - largely superseded by USB
- Card Security features
  - Read-only physical lock (not on microSD)
  - Reversible and non-reversible commands to disable writing
  - Card password – up to 16 bytes
  - DRM copy protection (rarely used feature): ~10% of SD capacity is a “protected area” not available to the user. Used to verify the identity of an application program.
  - The DRM protected area may be erased by reformatting. To avoid that can download the formatter from sdcard.org.

SD

- Electrical interface
  - Initially uses 3.3 volts. On command, SDHC, SDXC cards switch to 1.8 V operation
  - Initial clock speed should be 400 kHz when determining the card's capabilities after which the "default" speed of 25 MHz may be used
- SPI interface
  - SD cards have a built in SPI interface
  - This allows them to be used as non-volatile memory by embedded controllers without having to implement the full SD interface
- Error detection/correction
  - Read/write operations on the NAND flash technology used in SD cards may have errors therefore every operation includes a CRC (cyclic redundancy checksum)
  - If an error is detected the operation is repeated by controller until success

SD

- Storage format
  - Block character device
    - Data is read and written in blocks of fixed size
    - Blocks may be accessed in random order
  - The SD standard specifies these file formatting systems
    - SD – Up to 2 GB using FAT 12 and 16
    - SDHC – over 2GB-32GB using FAT32
    - SDXC – over 32GB-2TB using exFAT (Microsoft standard)

SD/MMC

References

https://www.sdcard.org/

- “Simplified” specs can be downloaded free
- Full specs require license fee

http://en.wikipedia.org/wiki/Secure_Digital

http://en.wikipedia.org/wiki/MultiMediaCard

http://en.wikipedia.org/wiki/CompactFlash

FAT File System

Introduction

- The File Allocation Table (FAT) File System was developed by Microsoft as a way of organizing data into files and directories on a magnetic disk.
- **Cluster**: The disk is divided into equal size units called **clusters**
- **File**: A **file** is an entity with a name, some attributes like timestamp, and a chain of clusters containing the file contents
- **Directory**: A **directory** is a special file containing a list of file names, their attributes, and their starting clusters
- History
  - 1979 – 8-bit FAT entries – allowed 2^8 clusters
  - 1981 – 12-bit FAT entries – MS-DOS 1.0 – allowed 2^12 clusters
  - 1984 – 16-bit FAT entries – MS-DOS 3.0 – allowed 2^16 clusters
  - 1996 – 32-bit FAT entries – MS-DOS 7.1 – allowed 2^32 clusters
- Today, the same FAT file system technology developed for magnetic disks is commonly repurposed for use on Flash devices like USB memory keys and SD Cards

FAT File System

Terms

- **Sector** – smallest unit of read/write storage access – 512 bytes == **block size**
- **Cluster** – smallest unit of file storage – a set of contiguous sectors – a power of 2 – commonly 32768 bytes for SD cards
- **File** – a chain of clusters
- **Directory**
  - a special kind of file containing a list of files and their starting clusters
  - A directory file may contain other directory files resulting in a tree structure of directories
- **File Allocation Table (FAT)** – contains an entry for every cluster:
  - Zero indicates an unallocated cluster
  - 0xFFFF indicates last cluster of a file (FAT16)
  - Any other value indicates next cluster in a file

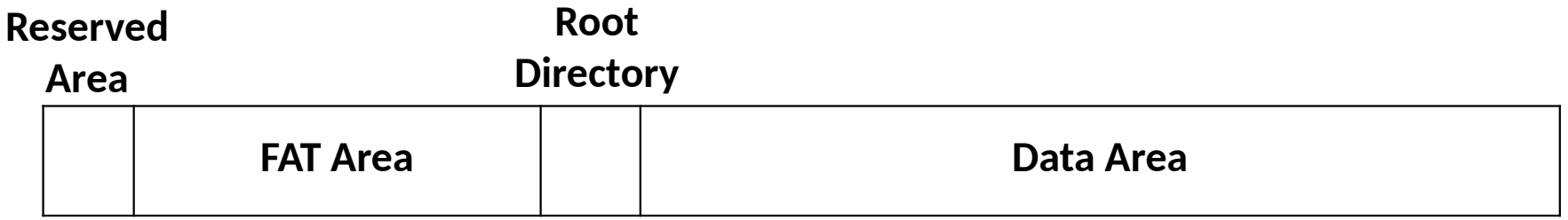
FAT File System

Terms cont'd

- **Volume**
  - A floppy disk, hard disk drive, USB memory stick, SD Card, etc. organized as a FAT file system
- **Boot Sector, BIOS Parameter Block (BPB), Zeroth Sector, Reserved Sector**
  - All these terms refer to the first sector (Sector 0) on a FAT volume which contains critical information to allow the data on the volume to be read and written
  - Single point of failure for FAT file system
- **Root Directory**
  - The file containing the starting directory of the directory tree of the volume
  - On FAT32 it is a file like any other, on FAT12 and FAT16 it is of fixed length
- **Partition**
  - (Volume) – division of a hard disk into multiple logical hard disks each with a file system
- **Standard FAT sizes – we'll focus on FAT32 since it is standard for SDHC**
  - **FAT12** – allows for 2^12 = 4096 clusters – used for floppy disks
  - **FAT16** – allows for 2^16 = 65,536 clusters – up to 2 GB
  - **FAT32** – allows for 2^32 = 4,294,967,296 clusters – up to 2 TB

FAT File System

Simple Volume Layout



FAT File System - Boot sector

Boot Sector bytes 0-35

Bytes	Purpose
0-2	Assembly code instructions to jump to boot code (mandatory in bootable partition)
3-10	OEM name in ASCII
11-12	Bytes per sector (512, 1024, 2048, or 4096)
13	Sectors per cluster (Must be a power of 2 and cluster size must be <=32 KB)
14-15	Size of reserved area, in sectors
16	Number of FATs (usually 2)
17-18	Maximum number of files in the root directory (FAT12/16; 0 for FAT32)
19-20	Number of sectors in the file system; if 2 B is not large enough, set to 0 and use 4 B value in bytes 32-35 below
21	Media type (0xf0=removable disk, 0xf8=fixed disk)
22-23	Size of each FAT, in sectors, for FAT12/16; 0 for FAT32
24-25	Sectors per track in storage device
26-27	Number of heads in storage device
28-31	Number of sectors before the start partition
32-35	Number of sectors in the file system; this field will be 0 if the 2B field above (bytes 19-20) is non-zero

FAT File System – Boot Sector

Boot Sector bytes 36-65

- Specific to FAT32

Bytes	Purpose
36-39	FATSz32 - Count of sectors occupied by one FAT
40-41	ExtFlags – flags for active and mirrored FAT
42-43	FSVer – version number
44-47	RootClus – cluster number of first cluster of the root directory
48-49	FSInfo – sector number of the FSINFO structure in the reserved area of the FAT32 volume. Usually 1.
50-51	BkBootSec – if non-zero, indicates the sector number in the reserved area of the volume of the copy of the boot record. Usually 6. No value other than 6 is recommended.
52-63	Reserved
64	DrvNum – drive number of the media
65	Reserved1

FAT File System – Boot Sector

Boot Sector bytes 66-89

- Specific to FAT32

Bytes	Purpose
66	BootSig – Extended boot signature (0x29). Indicates that the following 3 fields in the boot sector are present
67-70	VolID – Volume serial number.
71-81	VolLab – matches the 11-byte volume label recorded in the root directory
82-89	FilSysType – Always set to the string “FAT32 “ for FAT32 volumes

IMPORTANT: Additionally, for the boot sector of a FAT volume, byte 510 equals 0x55 and byte 511 equals 0xAA.

FAT file system

Contents of a directory entry structure

- 32 bytes long

FAT 32 Byte Directory Entry Structure

Name	Offset (byte)	Size (bytes)	Description
DIR_Name	0	11	Short name.
DIR_Attr	11	1	File attributes: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 ATTR_DIRECTORY 0x10 ATTR_ARCHIVE 0x20 ATTR_LONG_NAME ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that.
DIR_NTRes	12	1	Reserved for use by Windows NT. Set value to 0 when a file is created and never modify or look at it after that.
DIR_CrtTimeTenth	13	1	Millisecond stamp at file creation time. This field actually contains a count of tenths of a second. The granularity of the seconds part of DIR_CrtTime is 2 seconds so this field is a count of tenths of a second and its valid value range is 0-199 inclusive.
DIR_CrtTime	14	2	Time file was created.
DIR_CrtDate	16	2	Date file was created.
DIR_LstAccDate	18	2	Last access date. Note that there is no last access time, only a date. This is the date of last read or write. In the case of a write, this should be set to the same date as DIR_WrtDate.
DIR_FstClusHI	20	2	High word of this entry's first cluster number (always 0 for a FAT12 or FAT16 volume).
DIR_WrtTime	22	2	Time of last write. Note that file creation is considered a write.
DIR_WrtDate	24	2	Date of last write. Note that file creation is considered a write.
DIR_FstClusLO	26	2	Low word of this entry's first cluster number.
DIR_FileSize	28	4	32-bit DWORD holding this file's size in bytes.

FAT File System

Example file tree

- Contains 3 text files and 2 nested directories

```
E:\
|   File1.txt
|
\---Dir1
    |   File2.txt
    |
    \---Dir2
            File3.txt
```

FAT file system
Conceptual example

Root Directory (Cluster 0)

Name	Attributes	Cluster
File1	<attributes>	2
Dir1	<attributes>	4

Dir1

Name	Attributes	Cluster
.	<attributes>	4
..	<attributes>	0 (Root)
Dir2	<attributes>	8
File2	<attributes>	5

Dir2

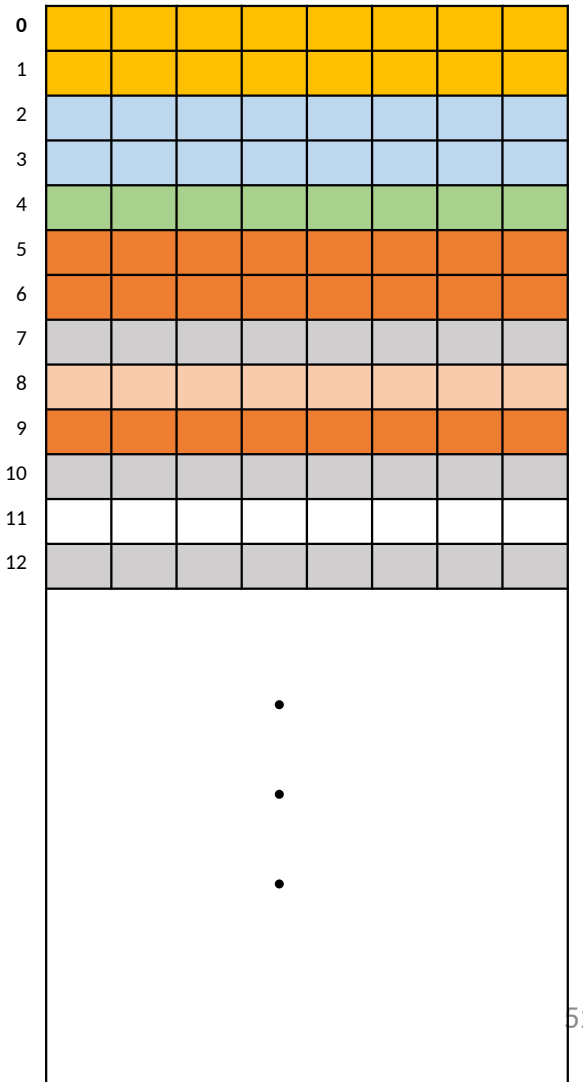
Name	Attributes	Cluster
.	<attributes>	8
..	<attributes>	4
File3	<attributes>	12

- First usable cluster is 2.
- 0 means unallocated cluster.
- FFFF means last cluster of a file.

File Allocation Table

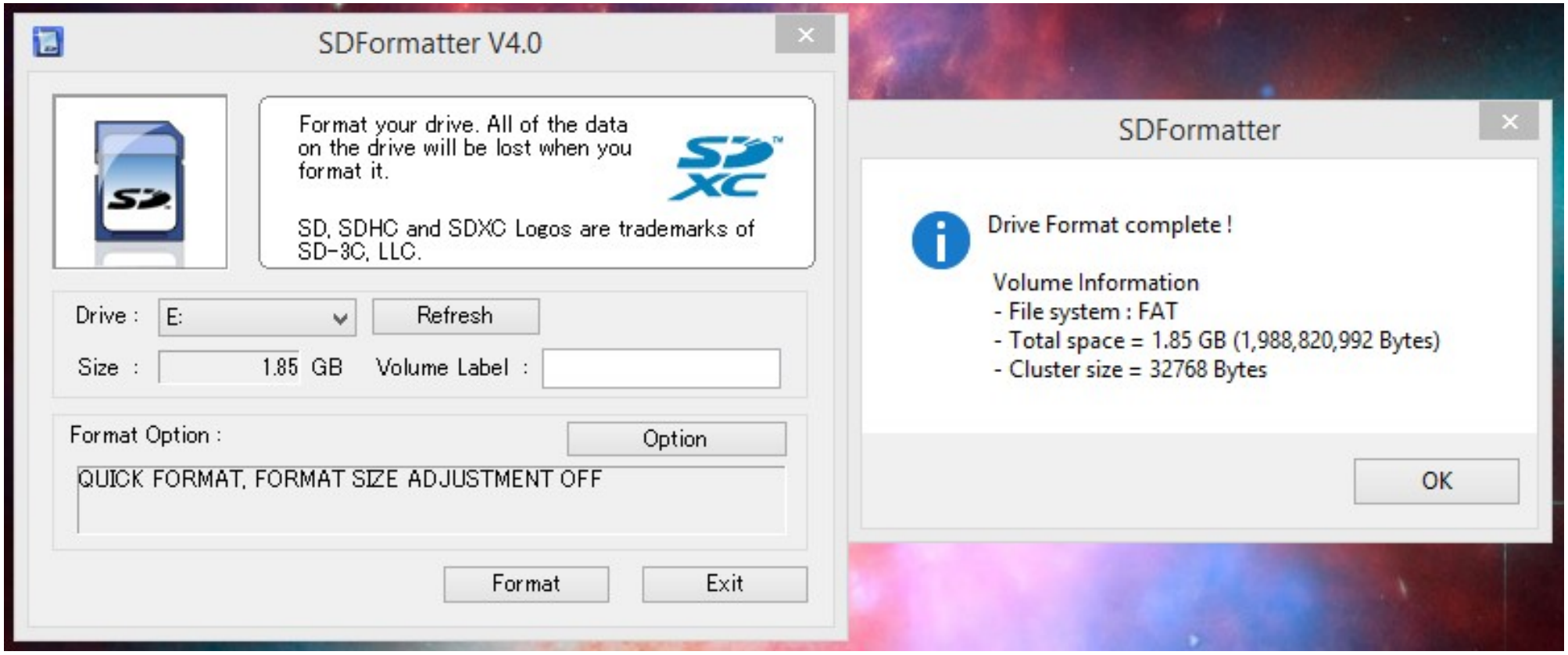
0	1
1	FFFF
2	3
3	FFFF
4	FFFF
5	9
6	FFFF
7	10
8	FFFF
9	6
10	FFFF
11	0
12	7

Cluster = 8 sectors (reality: usually 64 Sectors for FAT32)



FAT File System

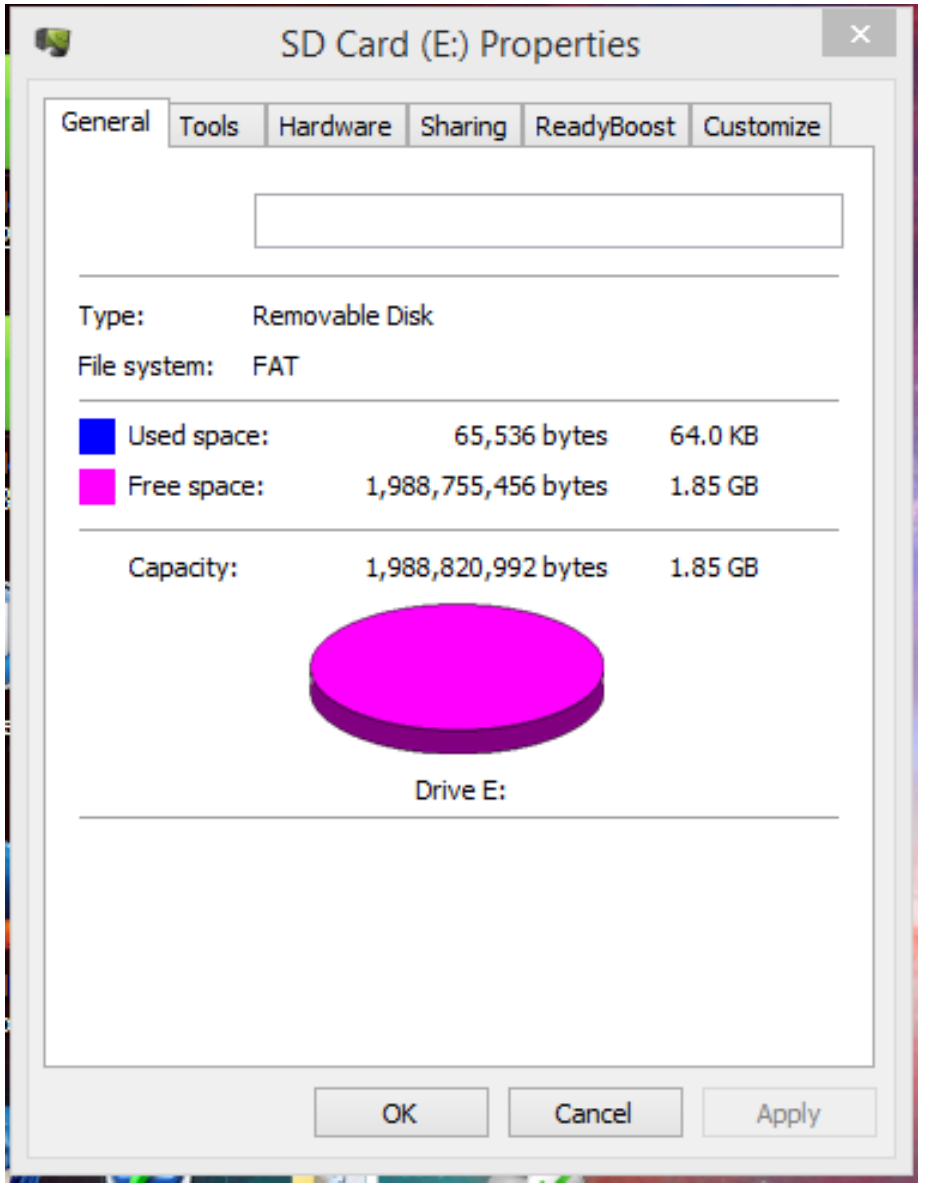
- Download SDFormatter.exe from sdcard.org
- Over time, FAT gets fragmented as files are deleted/created
- Formatting recreates a new FAT (old file data still present but cluster chains are lost)
- FAT (16) entries = DiskCapacity / ClusterSize = 1,988,820,992 / 32768 = 60,694



FAT File System

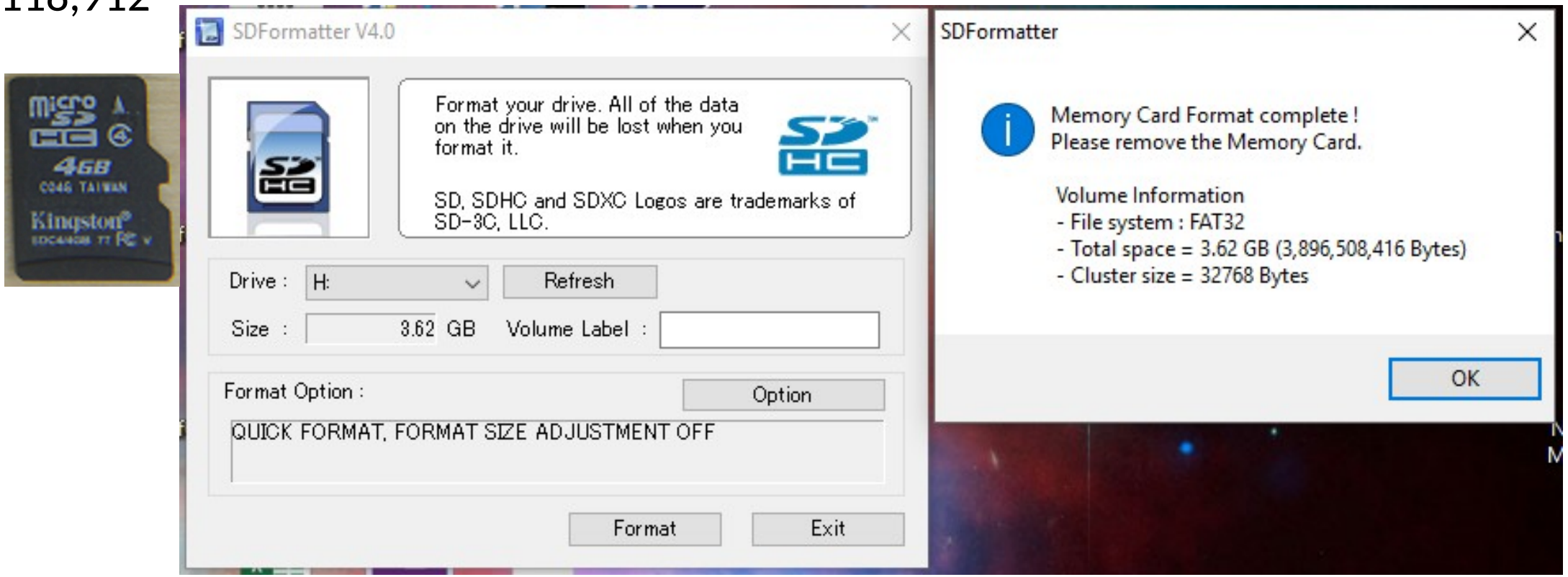
Formatted empty SDSC

2 clusters used



FAT File System

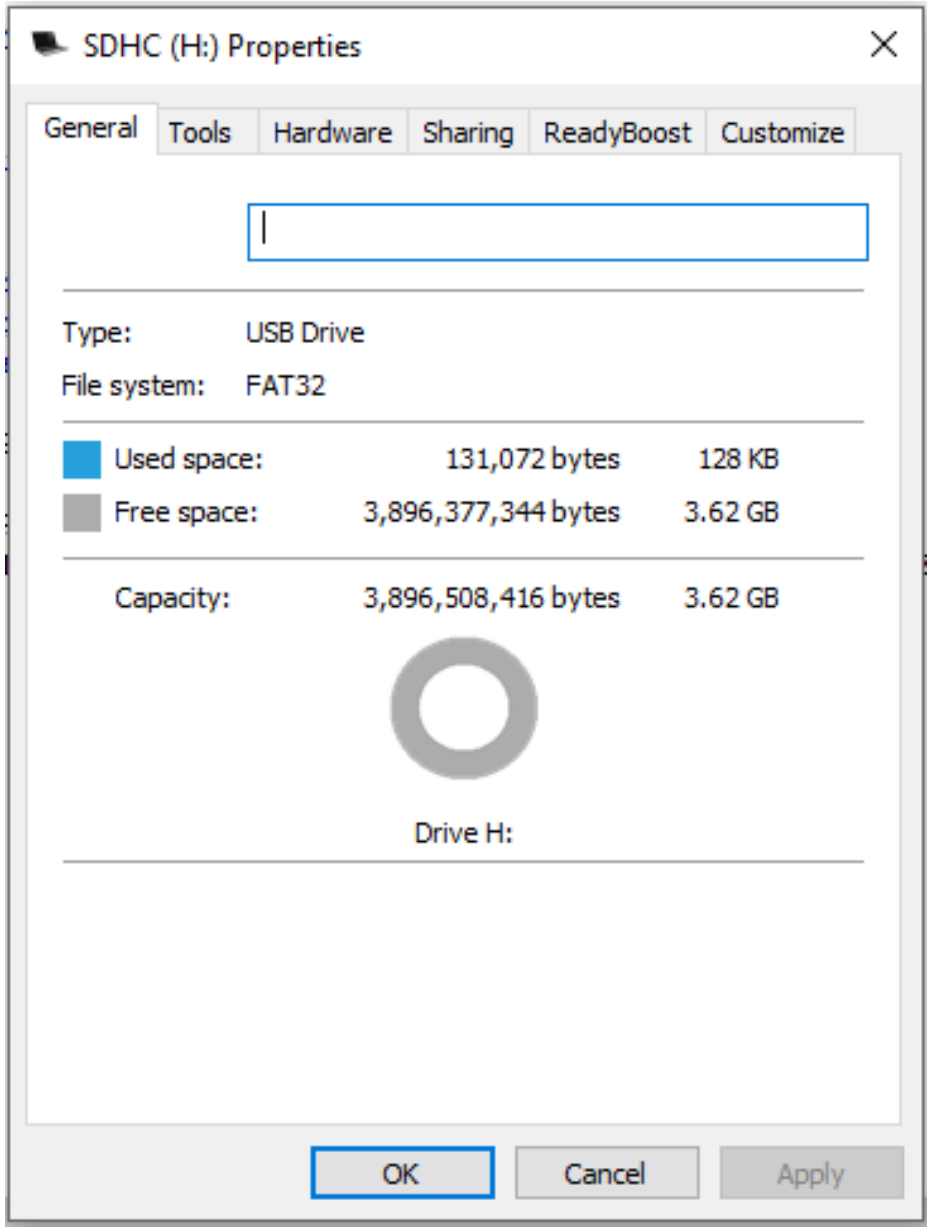
- Micro SDHC High Capacity 4 GB
- FAT32 entries = DiskCapacity / ClusterSize = 3,896,508,416 / 32768 = 118,912



FAT File System

Formatted empty SDHC

4 clusters used



FAT File System

FAT References

- FAT File System, Igor Kholodov, 2010
  - http://www.c-jump.com/CIS24/Slides/FAT/lecture.html#F01_0010_overview
- FatFs – Generic FAT File System Module, 2015
  - http://elm-chan.org/fsw/ff/00index_e.html
- File Allocation Table, Wikipedia, 2015
  - http://en.wikipedia.org/wiki/File_Allocation_Table
- Microsoft EFI FAT32 File System Specification, Microsoft, 2000
  - https://msdn.microsoft.com/en-us/windows/hardware/gg463080.aspx

Read a directory with Arduino SD library

- Sample code in Canvas:
  - “SD sample code.txt”

Read a directory with Arduino SD library

- Requires dynamic allocation:
  - uCOS memory partitions enabled in App/uCOS/os_cfg.h:

```
#define OS_MEM_EN                    1u
```

  - Limited dynamic allocation implemented in Arduino/SD/src/File.cpp