

VGAViewer

VISUALIZZAZIONE DI IMMAGINI SU UN' USCITA VGA TRAMITE STM32F407

ANDREA MONZANI

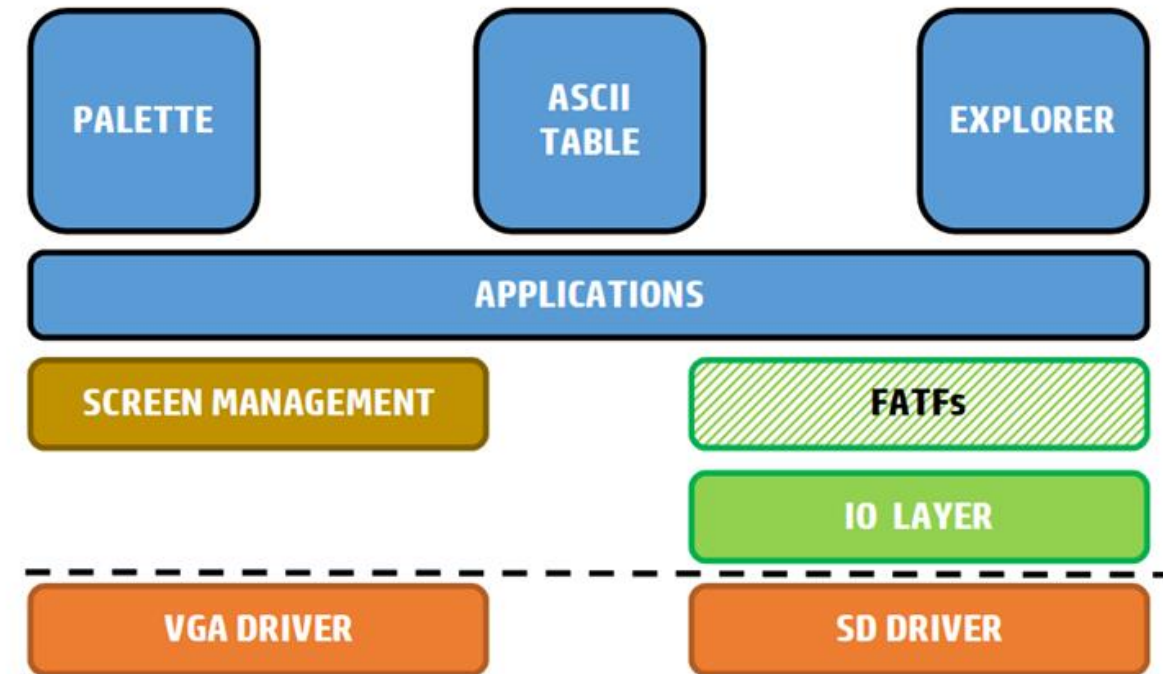
OBIETTIVI

A livello generale

- Connessione ad un monitor VGA
- Preparazione e visualizzazione di un frame buffer secondo le risoluzioni supportate
- Lettura di immagini BMP da una SD card

A livello software

- Creazione di due moduli che attraverso l' hardware dell' STM32F407 pilotano una porta VGA e una scheda SD
- Sviluppo di un layer intermedio per disegnare semplici elementi grafici (stringhe, rettangoli, ecc) a video
- Integrazione con FatFs (già messo a disposizione da CubeIDE)
- Preparazione di 3 applicazioni di esempio per valutare le prestazioni del sistema



Schema delle parti principali del progetto

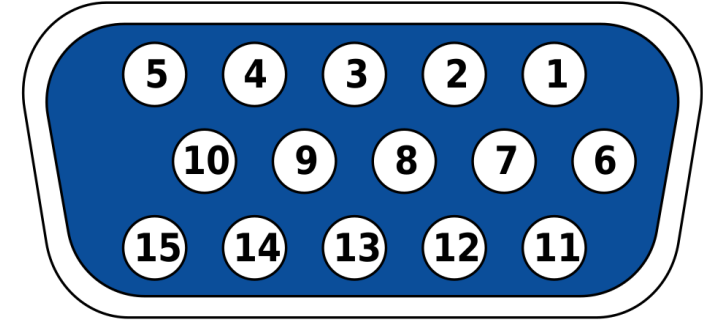
DRIVER
VGA

VGA – DI COSA ABBIAMO BISOGNO

PIN	DESCRIZIONE
1	Red video
2	Green video
3	Blue video
12	I ² C data
13	HSync
14	VSync
15	I ² C clock

Mappatura pin VGA

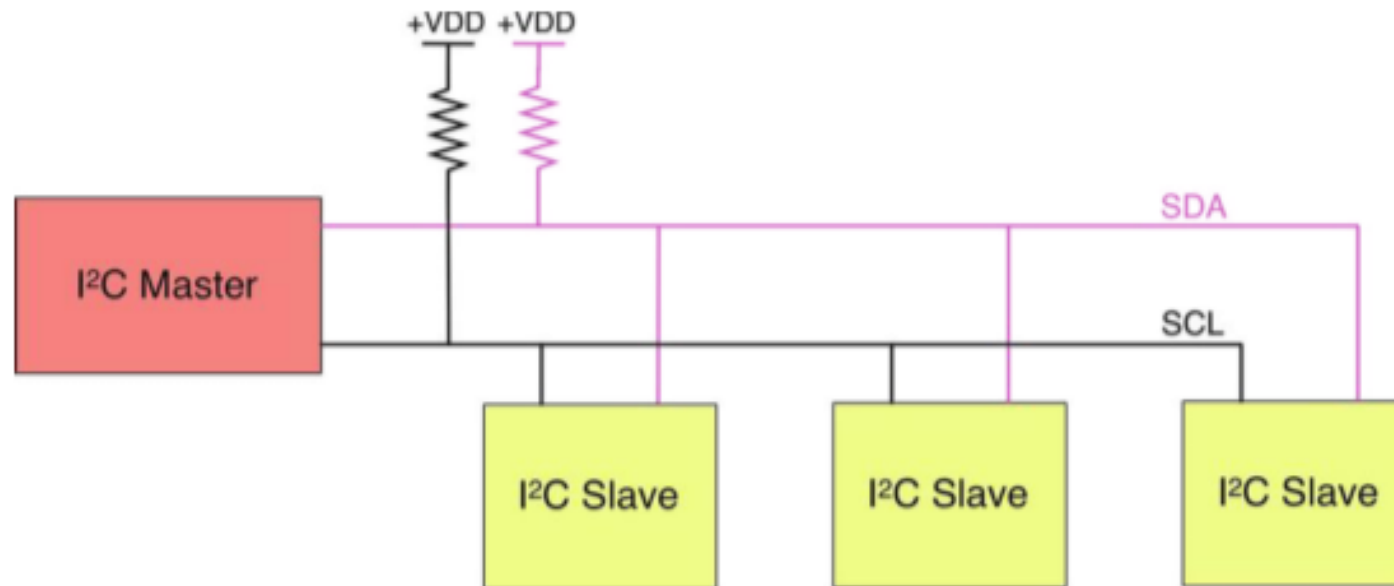
- 3 diversi segnali per i nostri colori nel range 0V – 0,7V
- 2 segnali per il syncing *dell'* immagine che (come vedremo) sono di tipo PWM con tempi dipendenti dal numero di pixel da disegnare
- Canale di comunicazione I²C con il monitor per la lettura dell' **EDID**



Schema connettore VGA

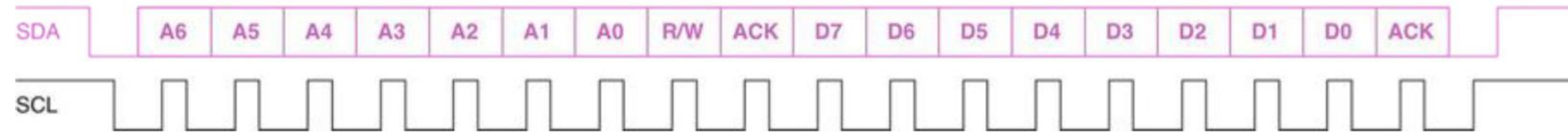
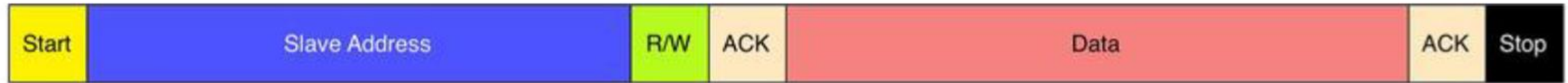
VGA – I²C

- Specifica *e protocollo* hardware, che offre una comunicazione sincrona, half duplex e multi slave
- Linea dati (SDA) e clock (SCL)
- Velocità dello standard mode tra i 100KHz e i 400KHz
- Ogni device è identificato da uno slave address univoco (7bit o 10 bit) nel bus
- Tutti i device devono essere in ascolto per rispondere al proprio slave address, quindi tutti devono lavorare alla stessa frequenza



Schema di un bus I²C

VGA – I²C PROTOCOL



Struttura di un messaggio I²C

- Transazioni sempre iniziate dal master, precedute sempre da una condizione di **start** e terminate con una condizione di **stop**
- Clock sempre generato dal master
- Inviati sempre almeno due frame: *address* e *data*, entrambi da 8 bit + ACK
- Può essere inviato un burst di data frame
- Il bit R/W indica chi dovrà riempire il frame di dati

VGA – LETTURA EDID

- *Extended Display Identification Data*
- Struttura che descrive le capacità di un dispositivo video
- 128 byte nella versione standard
- Contiene informazioni riguardo il costruttore, risoluzioni supportate, gamma, ed altre estensioni
- Necessario leggerla per verificare che una determinata risoluzione che vogliamo applicare sia supportata
- Assumeremo come disponibile e useremo la risoluzione **800x600@60Hz** (una delle più diffuse)
- Disponibile all' indirizzo 0x50 del bus I²C

```
Dumping Edid ...
Version: 1.3
Manufacturer: PTS
Product code: 5529
Week: 20
Year: 2004
Analog input
  Voltage levels: +0.7/0 V
  Blank to black not expected
  Separate sync supported
  Composite sync (on HSync) NOT supported
  Sync on green NOT supported
  Serrated VSync pulse (on Composite or SOG) NOT necessary
Basic timings
  640x480 @ 60Hz supported
  800x600 @ 56Hz NOT supported
  800x600 @ 60Hz supported
```

Esempio parziale di EDID letto da un monitor

BYTES		DESCRIPTION
20–24		Basic display parameters
20	Bit 6-5	Video white and sync levels, relative to blank: 00 = +0.7/−0.3 V 01 = +0.714/−0.286 V 10 = +1.0/−0.4 V 11 = +0.7/0 V (EVC)
35-37		Established timing bitmap
35	Bit 5	640×480 @ 60 Hz
	Bit 0	800×600 @ 60 Hz

Estratto EDID con alcuni dei bit che dobbiamo leggere

VGA – TIMING

General timing

Screen refresh rate	60 Hz
Vertical refresh	37.878787878788 kHz
Pixel freq.	40.0 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is positive.

Scanline part	Pixels	Time [μs]
Visible area	800	20
Front porch	40	1
Sync pulse	128	3.2
Back porch	88	2.2
Whole line	1056	26.4

Vertical timing (frame)

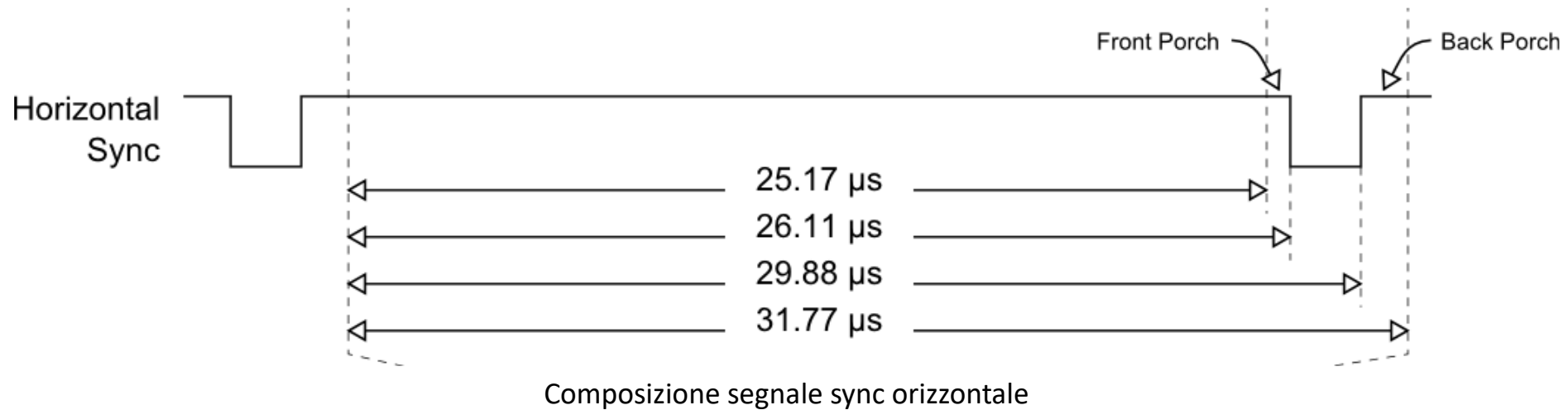
Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	600	15.84
Front porch	1	0.0264
Sync pulse	4	0.1056
Back porch	23	0.6072
Whole frame	628	16.5792

- Timing principale viene descritto attraverso la *pixel frequency*
- Area di disegno visualizzata corrisponde alla *visible area* come dimensioni
- Una volta generato il pixel clock con una certa precisione, la mappatura con il sistema di counter dei timer di STM32F407 è semplice
- HSI non è abbastanza preciso come clock interno, quindi è necessario utilizzare l' HSE abilitandolo dalla configurazione del *Reset and Clock Controller (RCC)*

Descrizione timing richiesti per la risoluzione 800x600@60Hz

VGA – SYNC SIGNALS



- Segnali a livello alto con l' impulso di sync a livello basso
- Segnale di sync verticale rispetta la struttura del sync orizzontale e viene incrementato alla fine di ogni back porch
- Durante la parte di porch e sync i segnali colore devono essere messi al livello *low*, detta anche blanking area
- Facilmente ottenibile da un timer con un canale configurato in PWM (il segnale generato sarà solo shifted perché il sync può essere solo alla fine o all' inizio)
- Possiamo costruire la catena di timing mettendo in cascata 3 timer attraverso il meccanismo di triggering interno

VGA – CONFIGURAZIONE TIMER

TIM1 Mode and Configuration

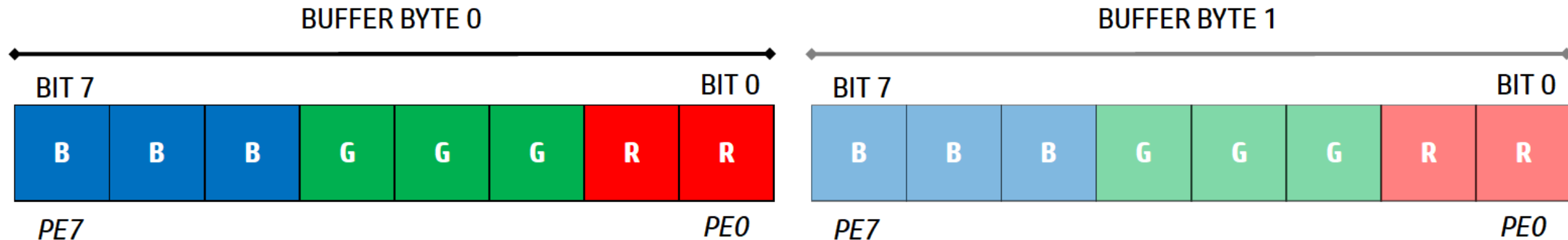
Mode	Configuration
Slave Mode: External Clock Mode 1 TIMER IS TRIGGERED FROM OUTSIDE	Reset Configuration
Trigger Source: ITR3 SOURCE OF THE TRIGGERING IS TIM4	NVIC Settings DMA Settings GPIO Settings
Clock Source: Disable	Parameter Settings User Constants
Channel1: PWM Generation CH1 HSYNC SIGNAL	Configure the below parameters :
Channel2: PWM Generation CH2 VSYNC CLOCK	Search (Ctrl+F)
Channel3: PWM Generation No Output VISIBLE LINE START IT	auto-reload preload: Disable
Channel4: PWM Generation No Output VISIBLE LINE END IT	Slave Mode Controller: ETR mode 1
Combined Channels: Disable	Trigger Output (TRGO) Parameters
<input type="checkbox"/> Activate-Break-Input	Master/Slave Mode (MSM bit): Disable (Trigger input effect not delayed)
<input type="checkbox"/> Use ETR as Clearing Source	Trigger Event Selection: Output Compare (OC2REF) ← SOURCE OF THE OUTPUT TRIGGER SIGNAL
	Break And Dead Time management - ...
	BRK State

Configurazione timer sync orizzontale

- Timer abilitato come slave su una source interna della scheda
- Il primo canale genera il segnale di sync che andrà al monitor
- Il secondo canale (solo per hsync) genera il tick di triggering per temporizzare il timer del sync verticale
- Gli altri due canali generano di continuo un interrupt all' inizio e alla fine dell' area visibile (sia per la linea che per il frame) . Vedremo poi come sarà gestito

VGA – FRAMEBUFFER ON STM32F407

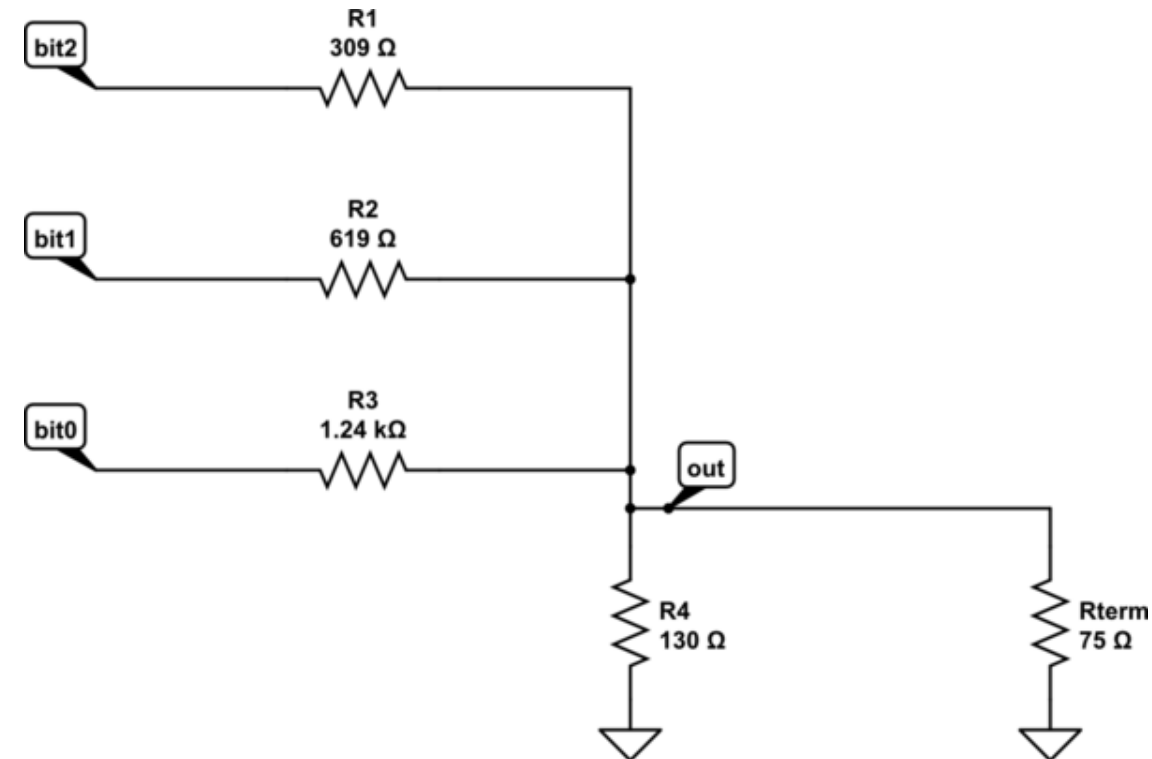
- Solo 128KB di SRAM
- Memoria FLASH è inutilizzabile per via delle limitazioni sulle scritture
- Buffer di disegno che possiamo utilizzare è al massimo di 400x300, **1 byte per pixel** (3 bit per verde e blu, 2 bit per rosso)
- Dimezzare il pixel clock e tutti i timing orizzontali per poi rispettare i limiti della risoluzione 800x600 che vogliamo usare
- Verticalmente disegneremo la stessa linea due volte
- *Per il momento* assumiamo che nella SRAM siano contenuti anche il nostro stack, heap e allocazioni statiche
- Essendo ogni singolo colore rappresentabile con un byte, possiamo ottimizzare e **disegnare 4 byte alla volta** in alcune situazioni



Configurazione framebuffer in memoria

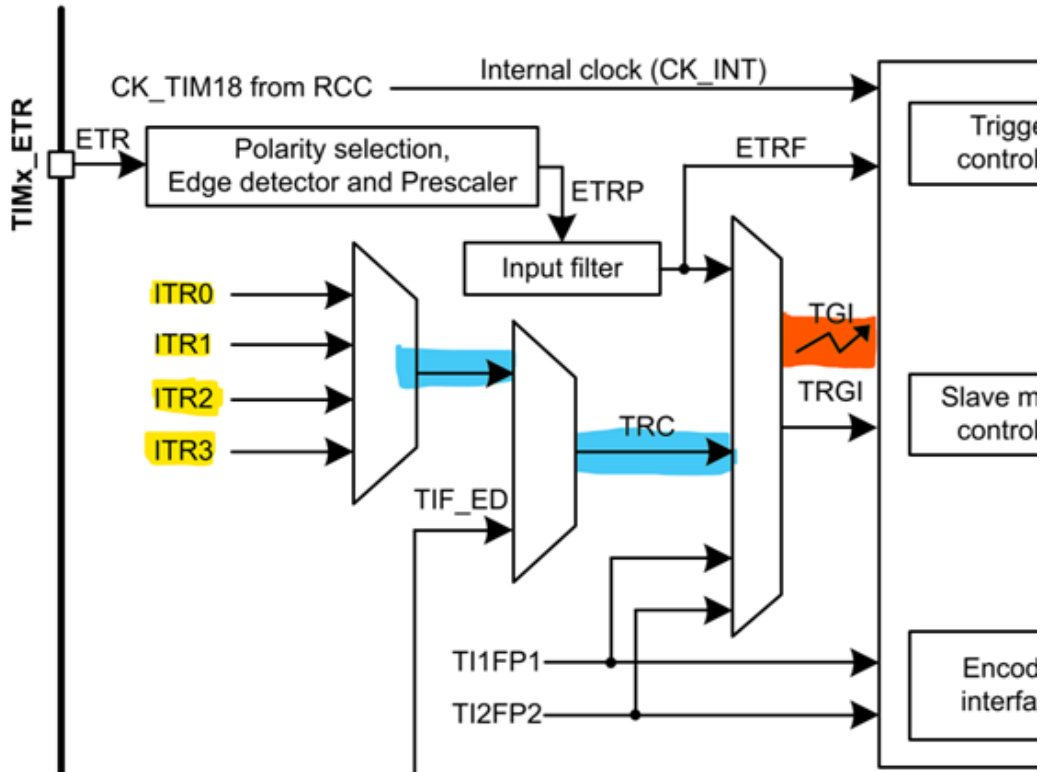
VGA – DAC

- Dobbiamo convertire i nostri 8 livelli di blu, verde e 4 di rossi in un segnale analogico
- STM32F407 dispone di un DAC interno ma ha diverse limitazioni
 - Risoluzione minima di 8 bit (non compatibile con i nostri requisiti)
 - Velocità massima di 1MS/s (aumentabile ma tramite un OpAmp esterno)
 - Due soli canali che possono lavorare in parallelo tramite DMA
- Dobbiamo ripiegare quindi su un DAC esterno tramite partitore resistivo
 - Semplice (con 3 bit non avremo comunque un buon risultato in termini di resa grafica)
 - Richiede solo delle resistenze



Configurazione partitore resistivo per 3 bit, *stackexchange*

VGA – DMA



Disponibilità di un evento di DMA/Interrupt per il TIM1 (hsync) quando temporizzato da una *Internal Trigger Source*

Requisiti

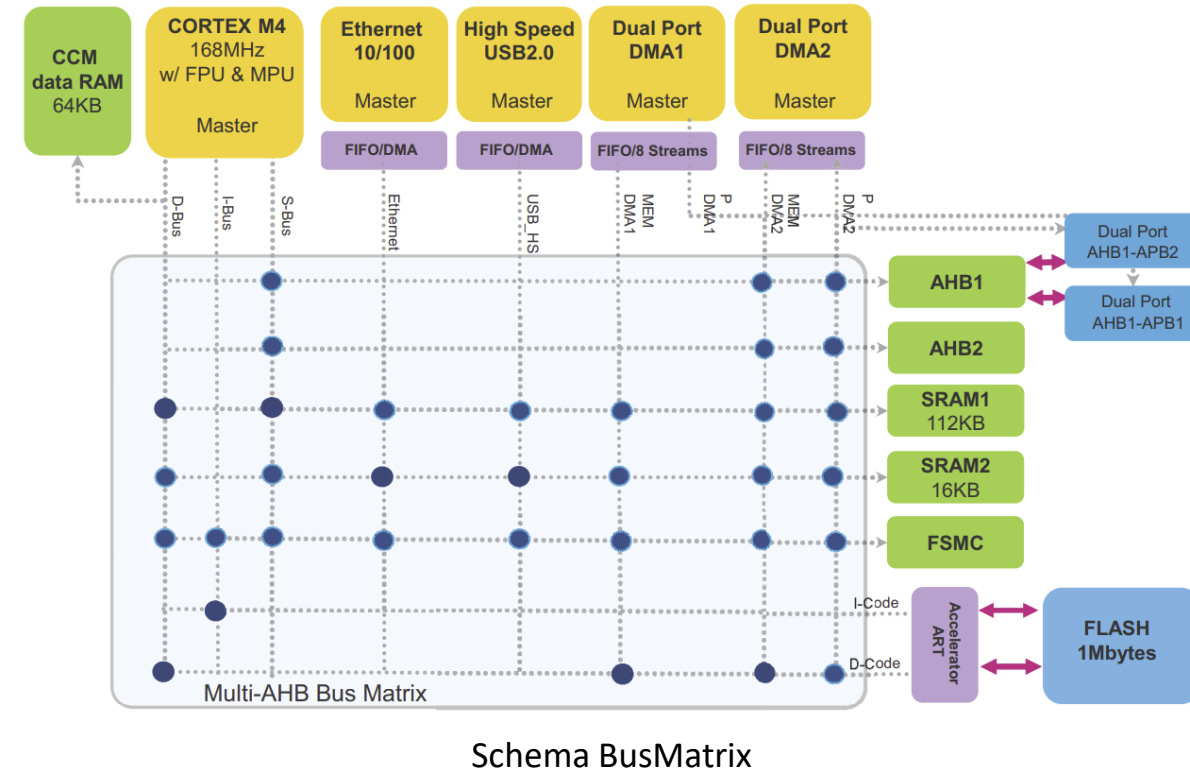
- DMA controller deve girare costantemente e inviare 400x300 bytes 60 volte al secondo **seguendo il pixel clock**
- Disegnare solo i dati nell' area visibile, disabilitato nelle aree di blanking

Implementazione

- Configurato in modalità memory to peripheral per trasferire i dati alla *GPIOE*
- Velocità stabilita dal pixel clock attraverso l' evento di triggering del timer riguardante l' hsync
- Abilitato e disabilitato nei gestori dell' interrupt dei nostri timer
- Deve lavorare in single mode (burst dei dati viaggia alla velocità dell' AHB quindi non rispetterebbe il pixel clock)

VGA – DMA, BUSMATRIX, OTTIMIZZAZIONI

- DMA è un master collegato al BusMatrix
- Deve per forza attraversarlo per ogni accesso alla memoria e per ogni scrittura sulla GPIO (non può passare direttamente dalla dual port) quindi dobbiamo capire come interagiscono
- Diversi problemi da analizzare
 - Tempo necessario per la lettura della memoria
 - Tempo necessario per la scrittura sullo slave
 - Caching via FIFO
 - **Contesa** sul BusMatrix



VGA – LATENZE BUSMATRIX

Description	Latency
t_{MA} : DMA memory port arbitration	1 AHB cycle
t_{MAC} : memory address computation	1 AHB cycle
t_{BMA} : bus matrix arbitration (when no concurrency) ⁽¹⁾	1 AHB cycle ⁽²⁾
t_{SRAM} : SRAM read or write access	1 AHB cycle

Description	Through bus matrix		DMA's direct paths
	To AHB peripherals	To APB peripherals	
t_{PA} : DMA peripheral port arbitration	1 AHB cycle	1 AHB cycle	1 AHB cycle
t_{PAC} : peripheral address computation	1 AHB cycle	1 AHB cycle	1 AHB cycle
t_{BMA} : bus matrix arbitration (when no concurrency) ⁽¹⁾	1 AHB cycle	1 AHB cycle	N/A
t_{EDT} : effective data transfer	1 AHB cycle ^{(2) (3)}	2 APB cycles	2 APB cycle
t_{BS} : bus synchronization	N/A	1 AHB cycle	1 AHB cycle

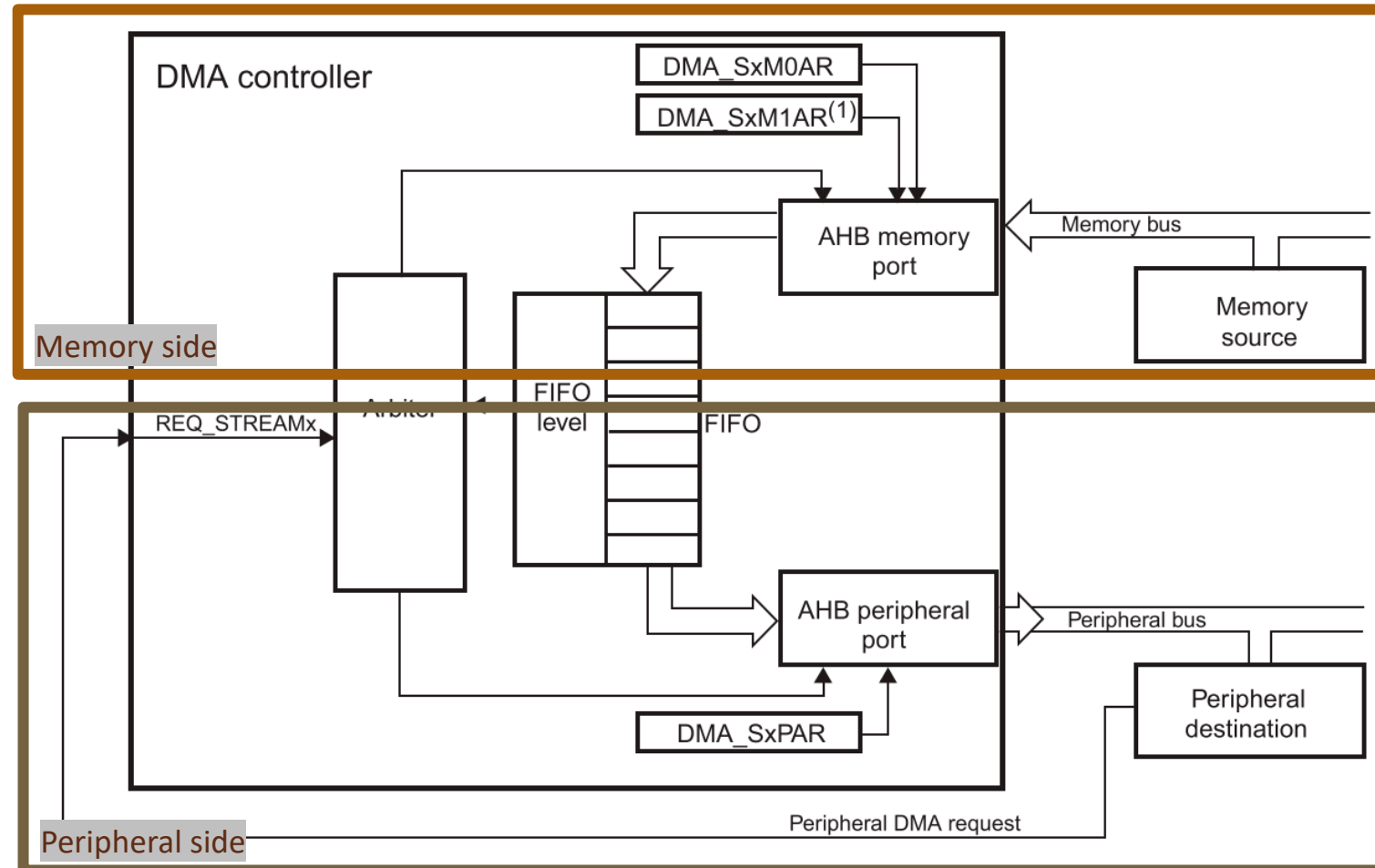
Latenze sul BusMatrix in cicli di clock dell' AHB

- Non abbiamo ulteriori ritardi sull' arbitraggio interno degli stream del DMA (abbiamo solo il canale dedicato al framebuffer)
- Leggere dalla memoria e scrivere sulla periferica sono operazioni con un costo ben definito (senza ulteriori contese)
- Il bus deve quindi girare ad almeno 4 volte la velocità del pixel clock per evitare ritardi nel disegno di pixel

VGA – FIFO

- Il DMA in direct mode senza FIFO deve leggere un byte dalla memoria ad ogni richiesta della periferica ed inviarlo alla GPIO
- Possiamo ottimizzare andando a recuperare «concorrentemente» i nostri dati

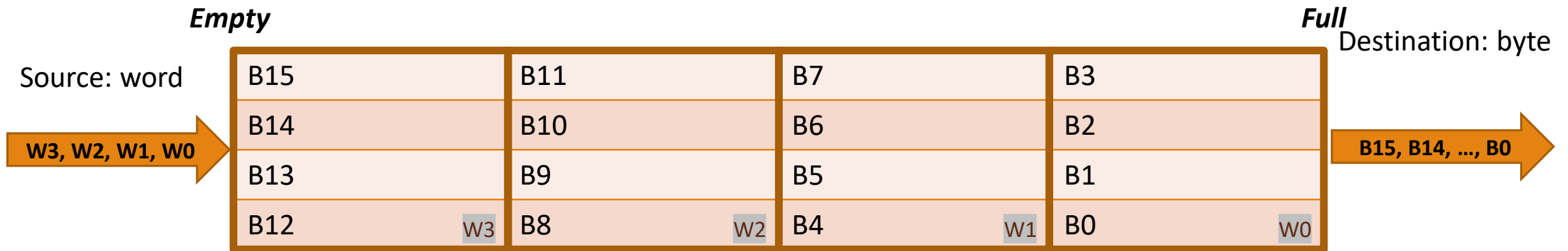
Legge i dati dalla memoria **byte per byte** (una volta sotto un certo limite) fino a riempire la FIFO



All' arrivo di una richiesta della periferica, prende **un byte** dalla FIFO e lo scrive sulla destinazione

VGA – FIFO UNPACKING

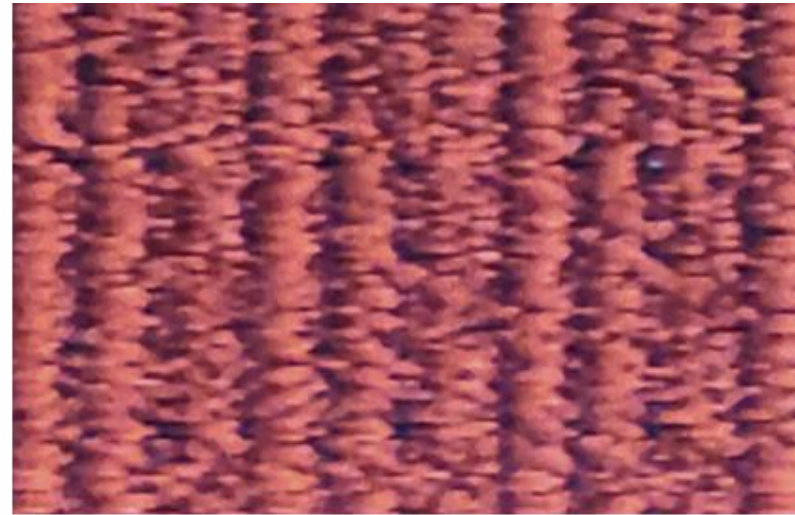
- Per ridurre il numero di accessi alla SRAM è possibile leggere word intere nella FIFO per poi inviare byte per byte ad ogni richiesta della periferica (unpacking)
- Riduciamo di 4 volte gli accessi



Schema unpacking tramite FIFO

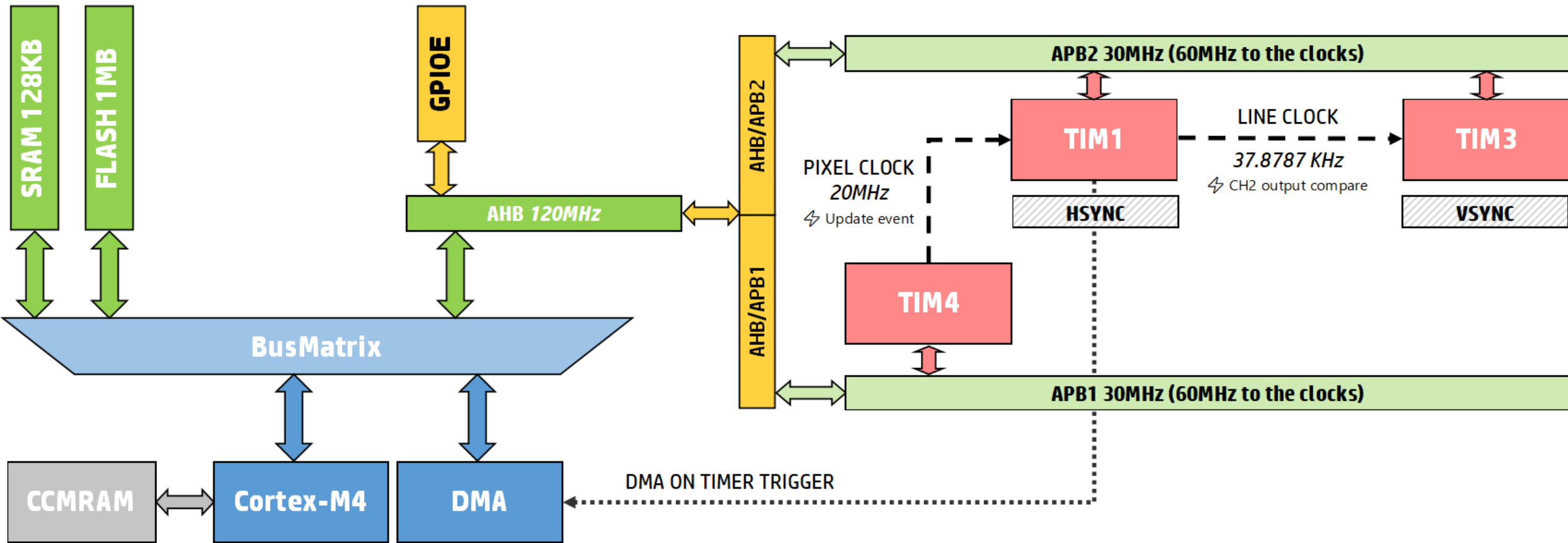
VGA – CONTESA

- BusMatrix implementa uno schema round robin per l'accesso agli slave. Abbiamo contesa (e quindi arbitraggio) quando due master diversi tentano di accedere allo stesso slave
- La latenza è quindi definita anche in base al numero di richieste attive da parte di altri master
- Possiamo contendere sull'accesso SRAM e sull'AHB
- Variabili del programma sono solitamente salvati in SRAM, ma possiamo spostare tutto nella CCM (*Core Coupled Memory*) per evitare di attraversare il BusMatrix al prezzo di avere meno memoria e non poter sfruttare il DMA
- **CPU blocca l'AHB per ridurre le latenze quando effettua operazioni multiple di load/store e ad ogni gestione di interrupt**
- **Problema** per quando usiamo FreeRTOS siccome si basa sugli interrupt **di SVCcall e PendSV**



Barre rosse senza contesa sul BusMatrix (sinistra), introducendo contesa sul BusMatrix (destra)

VGA – STRUTTURA TIMER E DMA





DEMO - PALETTE COLORI

FONT – COME E' SALVATO

- Dati del font sono stati esportati direttamente da Windows
- Ogni pixel è rappresentato da 65 livelli di colore (in scala di grigi)
- Per semplicità è stato generato il codice C contenente l' array di byte e salvato in un file. L' indirizzamento è in base al valore del carattere stesso
- Un singolo carattere è descritto tramite la struttura **GLYPHMETRICS**
- Viene specificato il minimo bounding box (chiamato black box), offset rispetto all' origine del disegno, incremento del puntatore di disegno
- Buffer è relativo solo al black box ed ogni linea è allineata alla word

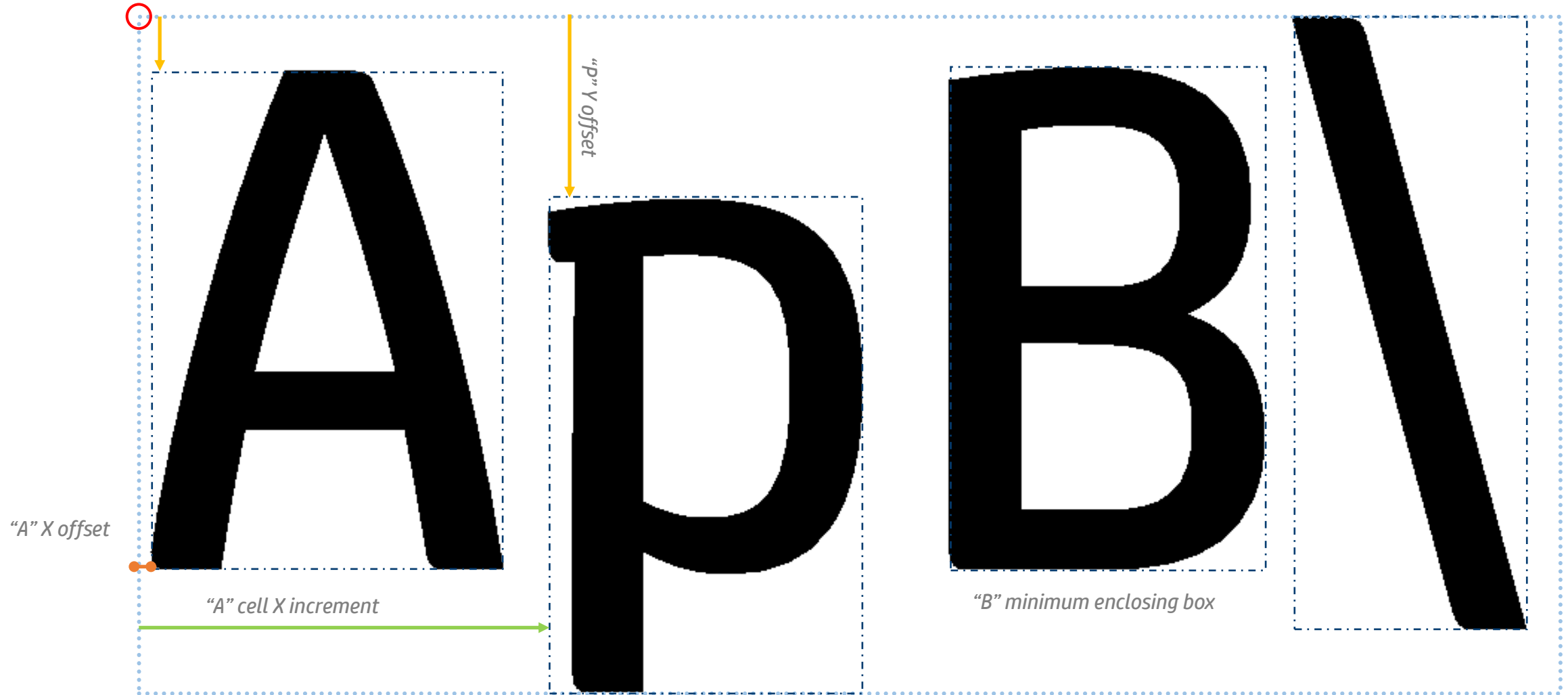
C++

```
typedef struct _GLYPHMETRICS {  
    UINT    gmBlackBoxX;  
    UINT    gmBlackBoxY;  
    POINT   gmptGlyphOrigin;  
    short   gmCellIncX;  
    short   gmCellIncY;  
} GLYPHMETRICS, *LPGLYPHMETRICS;
```

Definizione struttura *GLYPHMETRICS*

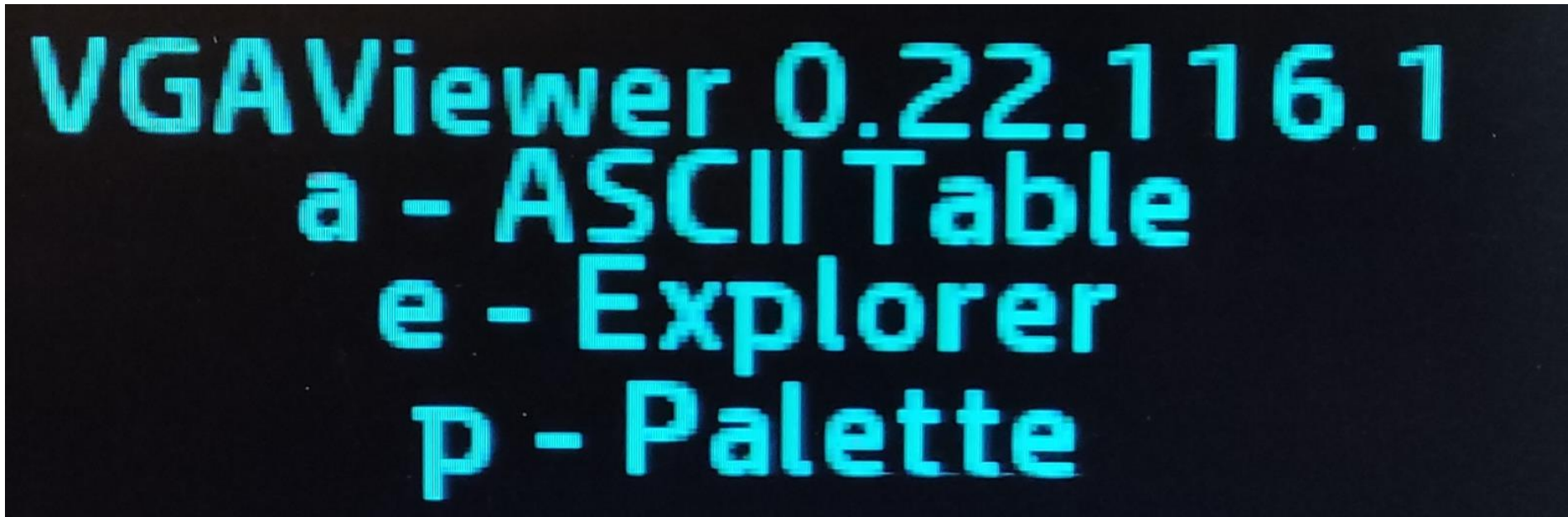
FONT – COME E' SALVATO

Origine del disegno (modificata per il progetto, normalmente è alla base delle lettere)



FONT – RENDERING

- Per disegnare una stringa, definiamo semplicemente un punto di origine e per ogni carattere:
 1. Carichiamo l' outline. Se non abbiamo nulla da visualizzare, ignoriamo il disegno
 2. Ci spostiamo dall' origine secondo i parametri indicati
 3. Tagliamo la black box per non uscire dai bordi dello schermo, se necessario
 4. Per ogni pixel di ogni linea, carichiamo il relativo livello e tramite alpha blending andiamo settare il colore
 5. Alla fine, in ogni caso, incrementiamo il puntatore di disegno usando le informazioni dell' outline

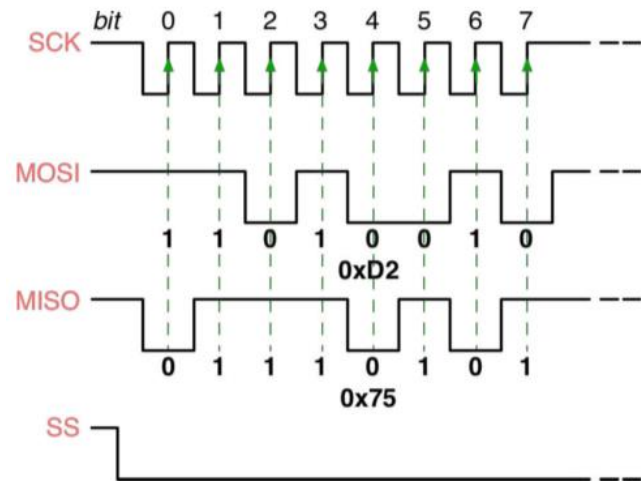


Risultato visualizzazione testo a schermo

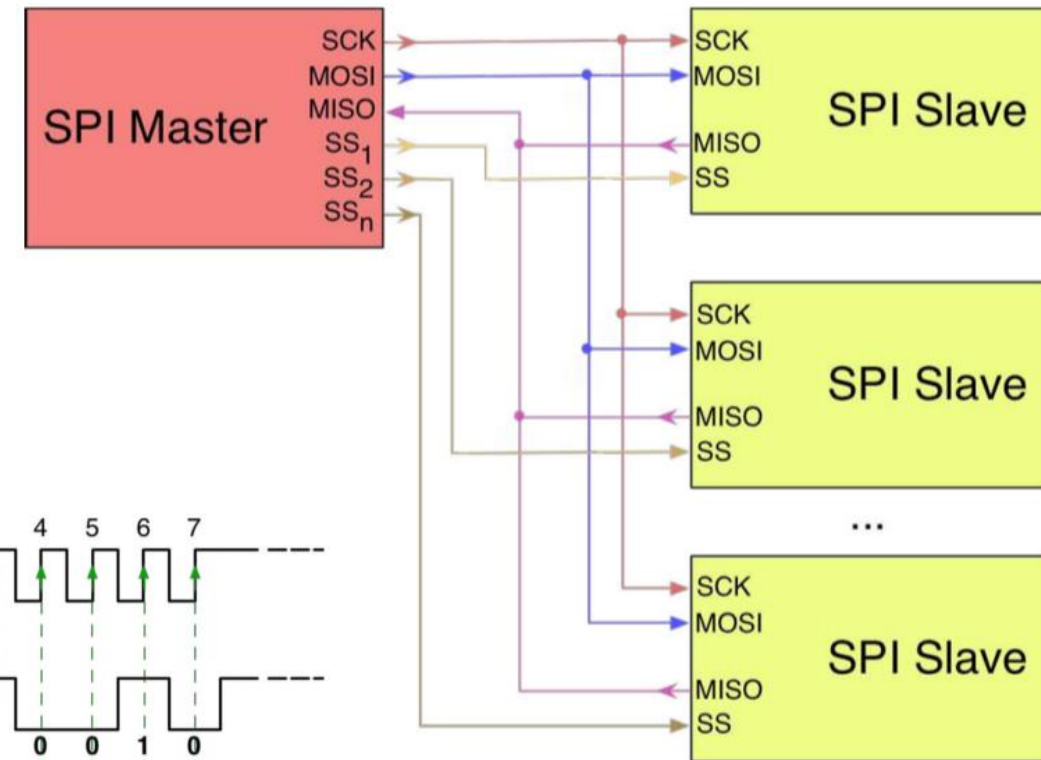
DRIVER
SD

SD – BUS SPI

- Comunicazione seriale, full-duplex, sincrona con altri device
- 2 canali dati, 1 di clock e uno di slave selection
- Velocità del clock variabile
- E' solo una specifica dei segnali e non forza nessun protocollo di comunicazione
- Linee dati e di clock sono solitamente usate in configurazione di pull-up
- Almeno un bit del primo byte dovrà essere 0 per capire che qualcuno sta iniziando a mandare qualcosa
- Quando riceviamo, essendo il bus sincrono, invieremo sempre un dummy byte (0xFF)



Esempio comunicazione full-duplex



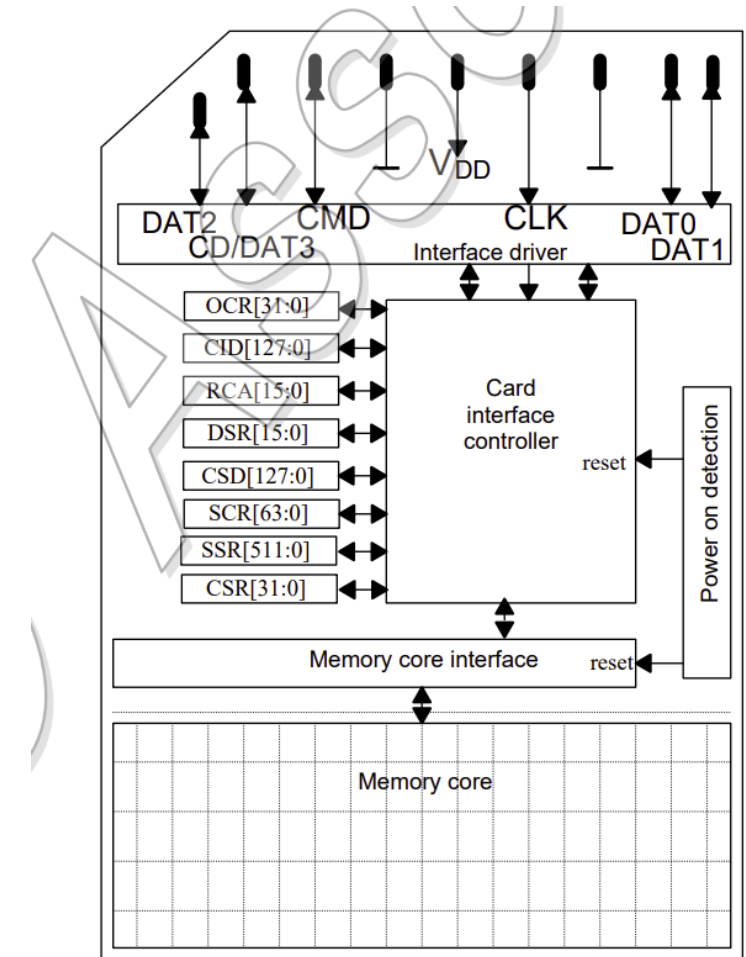
Schema bus SPI

SD – INFORMAZIONI PRINCIPALI

- Diversi protocolli di comunicazioni supportati, tra cui SPI
- Host invia dei comandi e l'interface controller della SD risponde
- Procedura specifica per il ciclo di accensione e inizializzazione
- Registri interni che descrivono alcuni parametri/limiti di funzionamento e le informazioni sul costruttore
- Memoria interna indirizzabile come una memoria normale; con un allineamento ben definito ma con alcune limitazioni

Pin #	SD Mode			SPI Mode		
	Name	Type ¹	Description	Name	Type ¹	Description
1	CD/DAT3 ²	I/O/PP ³	Card Detect/ Data Line [Bit 3]	CS	I ³	Chip Select (neg true)
2	CMD	I/O/PP	Command/Response	DI	I	Data In
3	VSS1	S	Supply voltage ground	VSS	S	Supply voltage ground
4	VDD	S	Supply voltage	VDD	S	Supply voltage
5	CLK	I	Clock	SCLK	I	Clock
6	VSS2	S	Supply voltage ground	VSS2	S	Supply voltage ground
7	DAT0	I/O/PP	Data Line [Bit 0]	DO	O/PP	Data Out
8	DAT1 ⁴	I/O/PP	Data Line [Bit 1]	RSV		
9	DAT2 ⁵	I/O/PP	Data Line [Bit 2]	RSV		

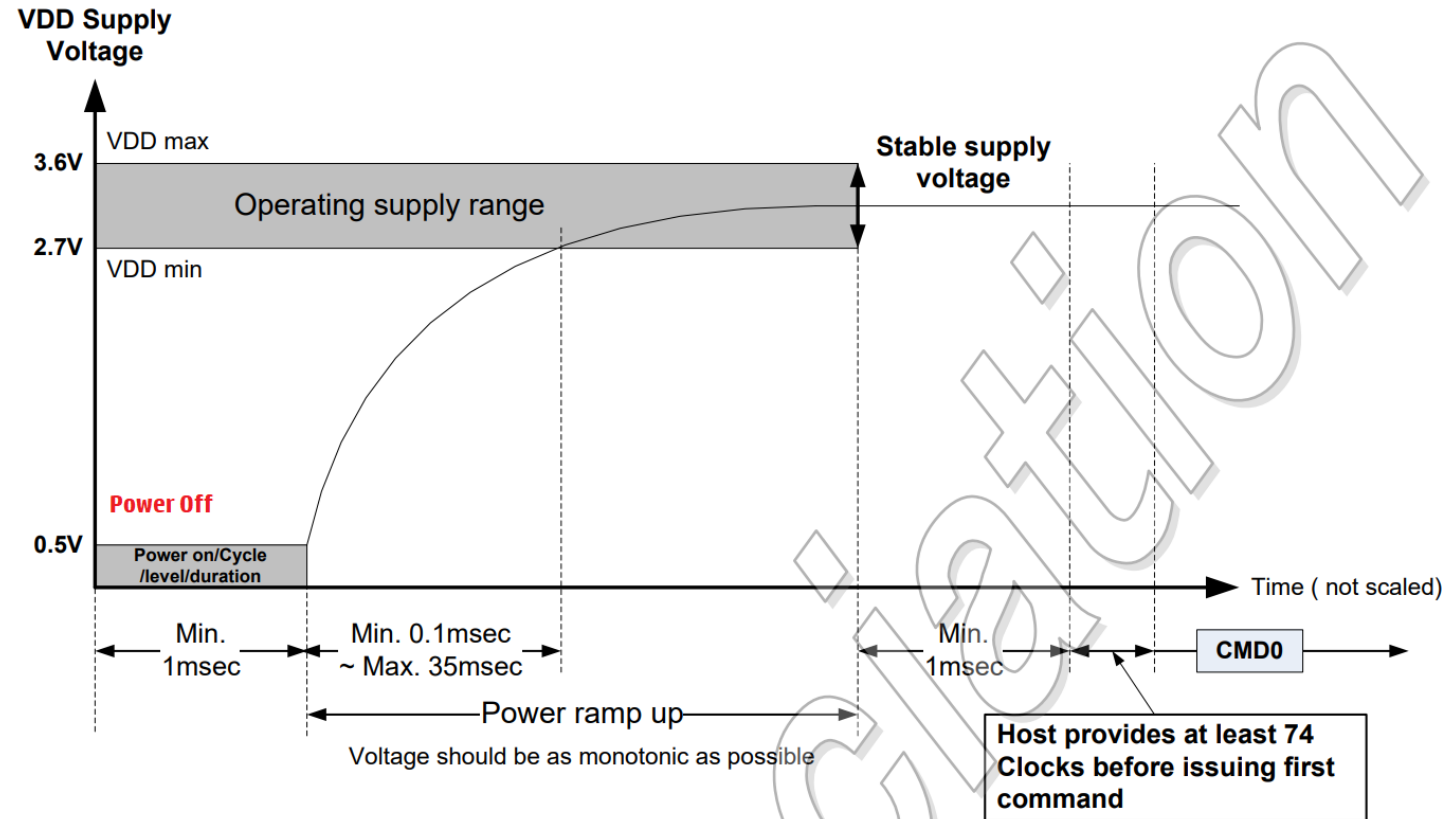
Mappatura pin nelle varie modalità



Schema interno scheda SD

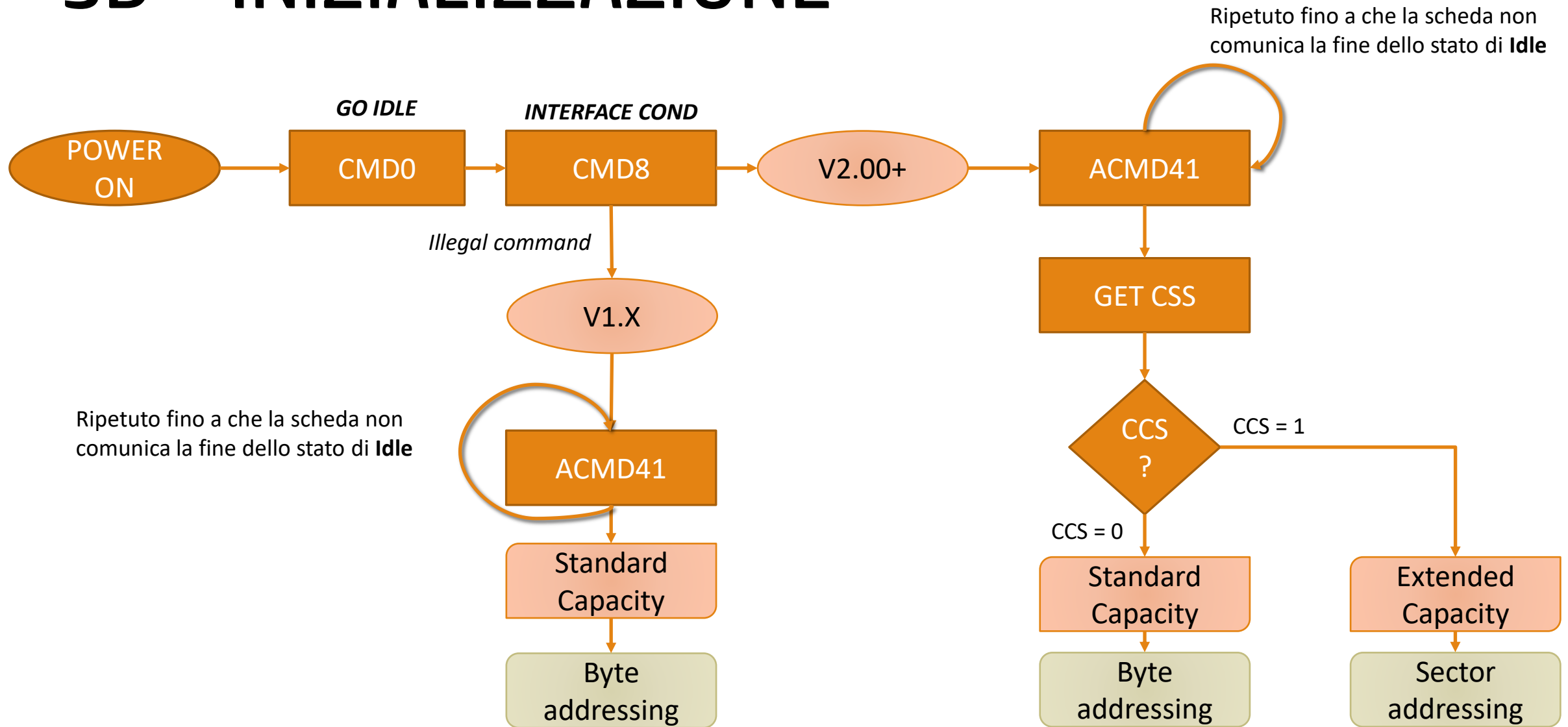
SD – POWER CYCLE

- Livello VDD deve essere a zero per almeno 1ms per spegnere la SD (le specifiche danno indicazioni anche per gli altri segnali)
- Livello VDD deve essere riportato al range operativo 2,7-3,6V nei limiti definiti (i nostri pin GPIO impiegano molto meno tempo secondo il datasheet ma non sembra influire sul power cycle)
- Attesa che il voltaggio si stabilizzi
- Clock SPI impostato tra i 100 e i 400 KHz
- **Invio di un certo numero di cicli di clock**
- Inizio schema di inizializzazione



Procedura di accensione di una scheda SD

SD – INIZIALIZZAZIONE



SD – FORMATO COMANDI

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

CMD INDEX	Argument	Abbreviation	Description
CMD0	/	GO_IDLE_STATE	Resetta memory card
CMD8	Supply voltage + check pattern	SEND_IF_COND	Invia la “interface condition”
CMD9	/	SEND_CSD	Invio CSD
CMD16	Block length	SET_BLOCKLEN	Imposta la dimensione di blocco per una card con Standard Capacity. Fissato a 512 per Extended Capacity
CMD17	Address (byte/blocco)	READ_SINGLE_BLOCK	Read a block of size selected by SET_BLOCKLEN

SD – OCR REGISTER (Operation Conditions)

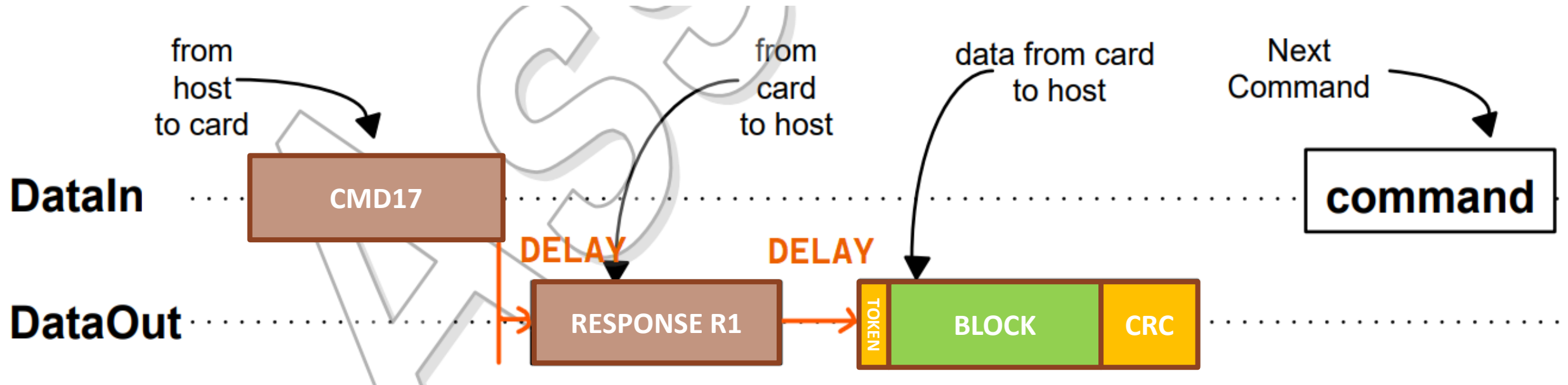
BIT POSITION	FIELD DEFINITION	COMMENTS
15	2,7 – 2,8	<i>VDD Voltage Window</i>
16	2,8 – 2,9	
17	2,9 – 3,0	
18	3,0 – 3,1	
19	3,1 – 3,2	
20	3,2 – 3,3	
21	3,3 – 3,4	
22	3,4 – 3,5	
23	3,5 – 3,6	
30	Card Capacity Status (CCS)	
31	Card power up status (busy)	

SD – CSD REGISTER (Card Specific Data)

NAME	VALUE	SLICE (bits)	COMMENT
CSD structure	00b 01b (Extended Capacity)	[127:126]	Valore dipendente dalla card capacity, indica semplicemente la versione
Max data transfer rate	32h (25Mhz) o 5Ah (50Mhz)	[103:96]	Velocità massima supportata per il clock SPI
Max read data block len	xh 9 (Extended Capacity)	[83:80]	Lunghezza massima blocco calcolata come $2^{\text{READ_BL_LEN}}$. Range valori è [9-11]
Partial block read	1b 0b (Extended Capacity)	[79:79]	Abilita la lettura di un blocchi più piccoli. La dimensione minima è di 1 byte
Read block misalignment	Xb 0 (Extended Capacity)	[77:77]	Possibile leggere oltre il limite di un blocco fisico

SD – Lettura di un blocco

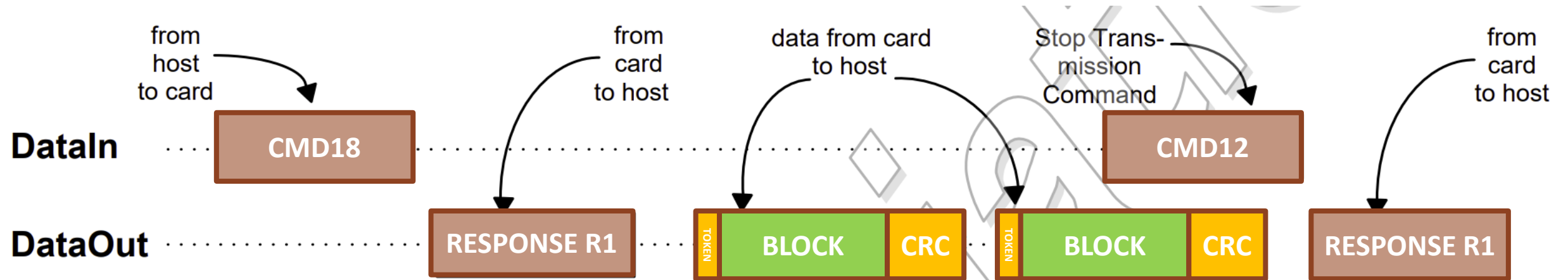
- Singolo blocco di dati può essere letto attraverso il comando **CMD17**
- Indirizzo del blocco calcolato in base alla Card Capacity (indirizzamento al byte o al settore)
- Riceviamo la risposta R1 relativa al comando
- Dobbiamo attendere un token (data o error)
- Leggiamo il blocco e poi abbiamo 2 byte di CRC



Schema lettura di un singolo blocco dalla scheda

SD – Lettura di più blocchi contigui

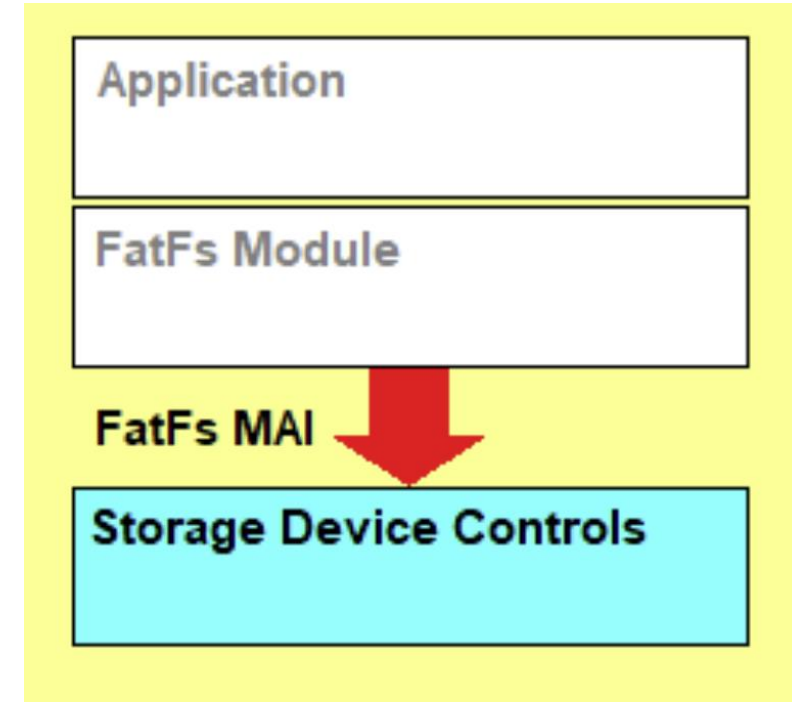
- Blocchi di dati contigui possono essere letti attraverso il comando **CMD18**
- Indirizzo del blocco iniziale calcolato in base alla Card Capacity (indirizzamento al byte o al settore)
- Riceviamo la risposta R1 relativa al comando
- In loop attendiamo token, leggiamo blocco e CRC fino a che non inviamo un comando di stop
- Evitiamo di mandare continuamente 6 byte di comando e 1 di risposta (più i tempi di delay)



Schema lettura di blocchi multipli dalla scheda

FatFs - Integrazione

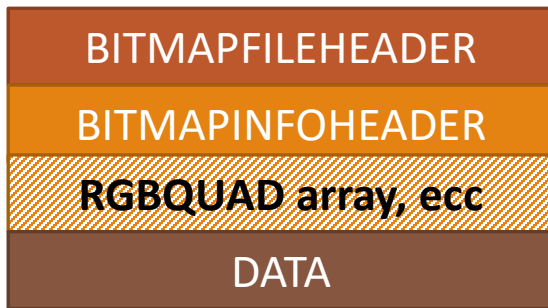
- FatFs mette a disposizione le più comuni funzioni di accesso ad un file system di tipo FAT
- Già predisposto nel CubeIDE
- Abilitate solo le funzione base di enumerazione di una directory e per la lettura di un file; non abbiamo scritture da fare sulla SD
- FatFs si interfaccia con un modulo chiamato *Media Access Interface* del quale dobbiamo implementare
 - **disk_initialize**: si deve connettere ad uno specifico device e metterlo in uno stato in cui è possibile poi eseguire delle letture (o scritture).
Per noi applicherà semplicemente un power cycle alla scheda SD e avvierà la procedura di inizializzazione tramite SPI
 - **disk_read**: deve leggere da uno specifico device un certo numero di **settori** a partire un certo offset (specificato sempre in numero di settori). Di conseguenza chiamerà semplicemente le funzioni messe a disposizione dal layer SD
- Dimensione del settore è fissata a 512 bytes, quindi non abbiamo bisogno di **ioctl** per ritornare la size corretta



Interfaccia FatFs

BPM

- Formato per il salvataggio di immagini bidimensionali. Supporta anche la compressione
- Diverse versioni nel corso del tempo, ci concentriamo sulla specifica Windows
- Composta da (nel caso non ci sia compressione/color palette):
 1. Header fisso per il file di 14 bytes contenente due byte iniziali per identificare la bitmap e 4 byte finali per l'offset a cui troveremo i dati
 2. Struttura fissa contenente informazioni sulle dimensioni e il colore
 3. Dati effettivi dell'immagine
- Solitamente l'origine della bitmap è lower-left (detta bottom-up DIB), quindi disegneremo le linee dal basso verso l'alto del nostro frame buffer



Bitmap structure

```
C++Copy  
  
typedef struct tagBITMAPFILEHEADER {  
    WORD    bfType;  
    DWORD   bfSize;  
    WORD    bfReserved1;  
    WORD    bfReserved2;  
    DWORD   bfOffBits;  
} BITMAPFILEHEADER, *LPBITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

Bitmap file header definition

BPM – INFO HEADER

- Di un file BMP ci interessa conoscere le dimensioni per applicare eventualmente un ridimensionamento
- Verificare che non sia stata applicata nessuna compressione
- Verificare che i colori siano salvati in modalità 24bpp
- Per ridimensionare possiamo applicare un semplice algoritmo come Nearest-neighbor
(non otterremo comunque risultati migliori applicando una interpolazione più complessa visto la resa grafica della nostra risoluzione e della profondità di colore)

C++

Copy

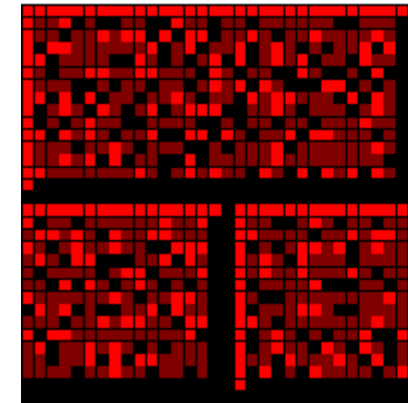
```
typedef struct tagBITMAPINFOHEADER {  
    DWORD biSize;  
    LONG  biWidth;    // The width of the bitmap, in pixels.  
    LONG  biHeight;   // The height of the bitmap, in pixels.  
    WORD  biPlanes;  
    WORD  biBitCount; // The number of bits-per-pixel.. If 24, no RGBQuad is present  
    DWORD biCompression;  
    DWORD biSizeImage;  
    LONG  biXPelsPerMeter;  
    LONG  biYPelsPerMeter;  
    DWORD biClrUsed;  // The number of color indexes in the color table that are actually used by the bitmap  
    DWORD biClrImportant; // The number of color indexes that are required for displaying the bitmap  
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

Bitmap header definition

Palette-index

```
row 0, scanline 31  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
row 1, scanline 30  00 00 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00 00  
row 2, scanline 29  11 11 01 19 11 01 10 10 09 09 01 09 11 11 01 90  
:  
:  
:  
row 31, scanline 0  99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 90
```

Pixel rectangle



Color-palette

0 Black
1 Dark red
2 Dark green
3 Dark yellow
4 Dark blue
5 Dark magenta
6 Dark cyan
7 Dark gray
8 Light gray
9 Light red
A Light green
B Light yellow
C Light blue
D Light magenta
E Light cyan
F White

Bitmap scanline example from Microsoft Docs, with color palette



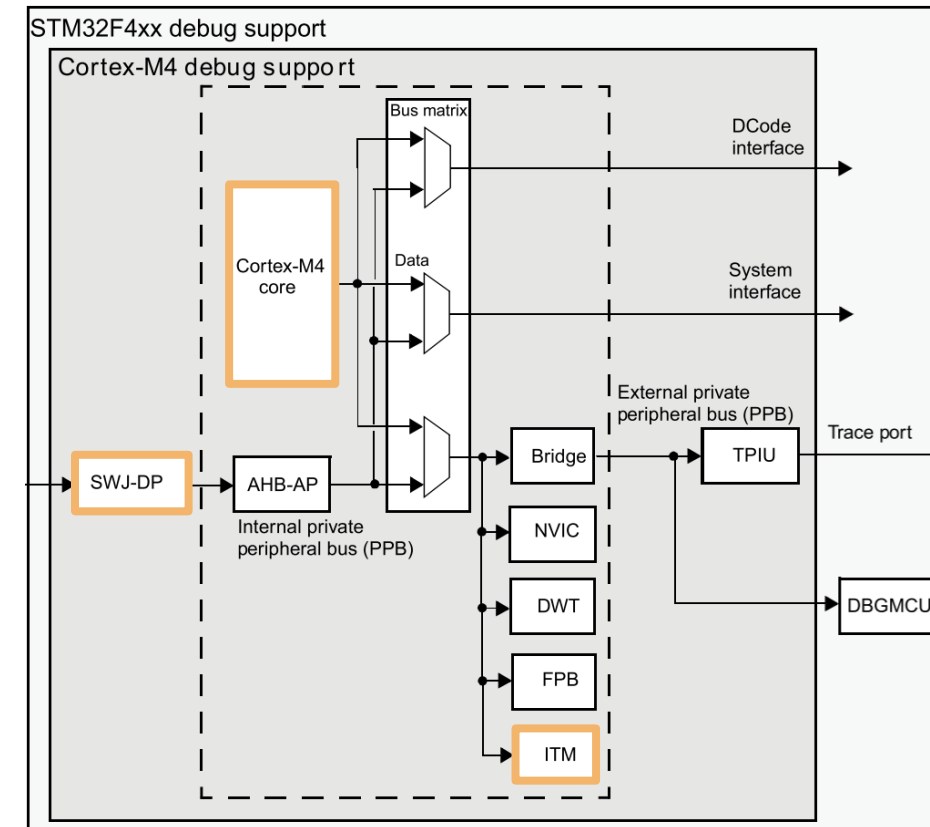
DEMO - EXPLORER

PRESTAZIONI – ITM

- L' Instrumentation Macro Cell abilita un supporto stile *printf()* per il debugging
- L' applicazione emette dei pacchetti dati etichettati con un certo timestamp attraverso la funzione *ITM_SendChar()*
- Disponibile tramite interfaccia di debug Serial Wire che deve essere preventivamente configurata

```
/**
 * \brief   ITM Send Character
 * \details Transmits a character via the ITM channel 0, and
 *           \li Just returns when no debugger is connected that has booked the output.
 *           \li Is blocking when a debugger is connected, but the previous character sent has not been transmitted.
 * \param [in]    ch  Character to transmit.
 * \returns       Character to transmit.
 */
STATIC_INLINE uint32_t ITM_SendChar (uint32_t ch)
{
    if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) &&          /* ITM enabled */
        (((ITM->TER & 1UL) != 0UL) &&                          /* ITM Port #0 enabled */
        ))
    {
        while (ITM->PORT[0U].u32 == 0UL)
        {
            __NOP();
        }
        ITM->PORT[0U].u8 = (uint8_t)ch;
    }
    return (ch);
}
```

Implementazione CMSIS della funzione ITM_SendChar



Schema interfaccia di debug

PRESTAZIONI – RIEMPIMENTO SCHERMO

- Se l'operazione risulta troppo lenta rispetto al disegno di un singolo frame (16 ms), è più facile notare a schermo il disturbo
- Disegnando un byte alla volta, abbiamo un tempo di circa 80 ms
- Disegnando 4 byte alla volta, otteniamo un incremento di circa 4 volte della velocità
- Non è sempre possibile ottimizzare
 - L'indirizzo del pixel deve essere allineato alla word
 - Dobbiamo essere sicuri di scrivere lo stesso colore (non applicabile con alpha ad esempio)
 - I pixel che stiamo scrivendo devono essere adiacenti
 - Pensato per il riempimento di grandi aree

ITM PACKETS – 1 BYTE DRAW

Index	Type	Data	Cycles	Time(s)
0	ITM Port 0	100 DRAW START	3079945330	25,666211 s
1	ITM Port 0	68 DRAW END	3089563325	25,746361 s
2	ITM Port 0	100	3089563402	25,746362 s
3	ITM Port 0	68	3099181322	25,826511 s
4	ITM Port 0	100	3099181397	25,826512 s
5	ITM Port 0	68	3108799322	25,906661 s

ITM PACKETS – 4 BYTES DRAW

Index	Type	Data	Cycles	Time(s)
2	ITM Port 0	100 DRAW START	163760002	1,364667 s
3	ITM Port 0	68 DRAW END	166621665	1,388514 s
4	ITM Port 0	100	166621744	1,388515 s
5	ITM Port 0	68	169483272	1,412361 s
6	ITM Port 0	100	169483351	1,412361 s
7	ITM Port 0	68	172345007	1,436208 s

Tempi riempimento scrivendo 1 byte alla volta o 4 byte alla volta

PRESTAZIONI – LETTURA DA SD

- Velocità effettiva di lettura (di un singolo blocco) è data principalmente dalla velocità del bus SPI (15 *Mhz*) nel nostro caso, ignorando i tempi necessari per l'invio del comando e l'attesa della risposta
- Overhead del nostro software (ignorando calcolo del CRC e controllo degli errori)
 - Non possiamo usare il DMA siccome il buffer gestito da FatFs è in CCM; potremmo spostarlo nel poco spazio rimanente della nostra SRAM ma dovremmo comunque poi attendere la fine della transazione (non escludendo neanche le latenze sul BusMatrix)
 - Leggendo direttamente in polling, dobbiamo sempre controllare i vari flag di lettura e scrittura del nostro registro dati
- Tempo di lettura teorico: 0.273 *ms*, effettivo: ~0.650 *ms*

```
PerformByteTransaction:
080032cc: ldr    r3, [pc, #20]    // (0x80032e4 <PerformByteTransaction+24>)
080032ce: ldr    r3, [r3, #0]     // Loading _spiInstance address in r3
080032d0: str    r0, [r3, #12]    // Storing r0 (byte parameter) in r3 + 12 [SPI Data Register]
+--> 080032d2: ldr    r2, [r3, #8]     // Loading r3 + 8 (SPI Status Register) value into r2
|      080032d4: and.w  r2, r2, #3       // We need to check Tx event and Rx event
|      080032d8: cmp    r2, #3
+--- 080032da: bne.n  <PerformByteTransaction+6> // Loop waiting
080032dc: ldr    r0, [r3, #12]    // Tx and Rx completed; We need to read Data Register again
080032de: uxtb   r0, r0           // Zero extend byte
080032e0: bx     lr              // Return to caller. Read byte will be in r0
```

Disassembly codice ottimizzato necessario per leggere un byte da SPI

Data	Cycles	Time(s)
98 START READ	1313975986	10,949800 s
66 END READ	1314053147	10,950443 s
98	1314561583	10,954680 s
66	1314638001	10,955317 s
98	1315170318	10,959753 s
66	1315247623	10,960397 s

Tempi lettura calcolati tramite ITM

GRAZIE PER L'ATTENZIONE

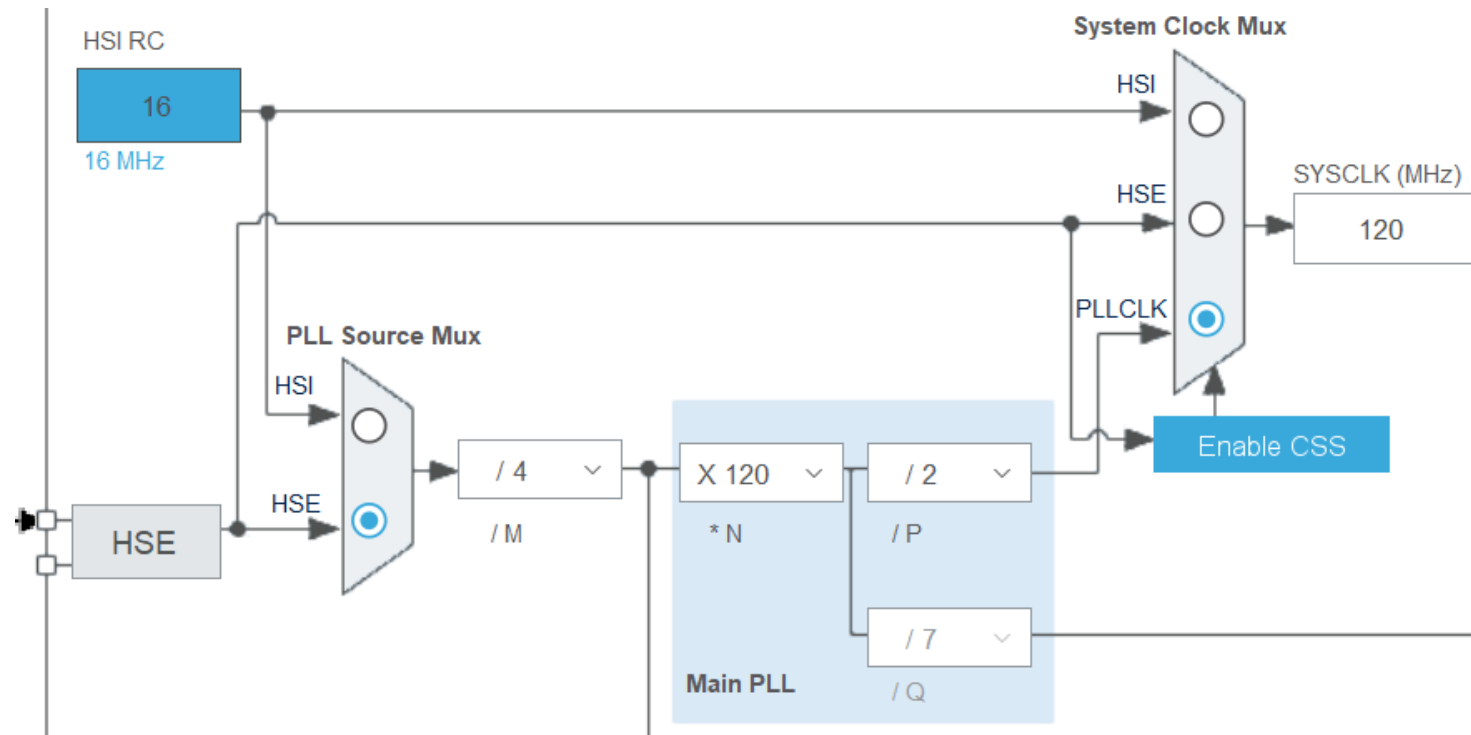
RIFERIMENTI

1. <https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html#documentation>; **STMF407 Documentation**
2. <https://github.com/SharathN25/STM32F407-Discovery/blob/master/README.md>; **Discovery board introduction**
3. https://en.wikipedia.org/wiki/Extended_Display_Identification_Data; **EDID**
4. https://en.wikipedia.org/wiki/Display_Data_Channel; **Display Data Channel**
5. <https://web.mit.edu/6.111/www/labkit/vga.shtml>; **VGA Signals**
6. <http://tinyvga.com/vga-timing>; **VGA Timings**
7. <https://docs.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-glyphmetrics>; **GLYPHMETRICS**
8. <https://docs.microsoft.com/en-us/windows/win32/gdi/bitmaps>; **BITMAPS**
9. <https://www.sdcard.org/>; **SD Association Website**
10. http://elm-chan.org/docs/mmc/mmc_e.html; **How use SD card in SPI mode**
11. http://elm-chan.org/fsw/ff/00index_e.html; **FatFs**

ULTERIORI INFORMAZIONI
















CLOCK SECURITY SYSTEM

- Attivato tramite software
- Monitora l' HSE e in caso di errore genera un Non-Maskable Interrupt
 - Bisogna pulire manualmente il relativo bit nel registro *RCC_CIR*
- Essendo l' HSE usato come sorgente del PLL, quest' ultimo verrà disabilitato e verrà riabilitato l' HSI



Configurazione HSE nel progetto, con possibilità di abilitare il CSS

CONFIGURAZIONE MEMORIA

Name	Run address (VMA)	Load address (LMA)	Size
▼  FLASH	0x08000000		1024 KB
>  .text	0x08000190	0x08000190	54,27 KB
>  .rodata	0x0800daa0	0x0800daa0	19,52 KB
>  .data	0x10000000	0x080128c0	2,77 KB
>  .isr_vector	0x08000000	0x08000000	392 B
 .ARM	0x080128b0	0x080128b0	8 B
>  .init_array	0x080128b8	0x080128b8	4 B
>  .fini_array	0x080128bc	0x080128bc	4 B
 .preinit_array	0x080128b8	0x080128b8	0 B
▼  RAM	0x20000000		128 KB
▼  .ram_data	0x20000000	0x20000000	128 KB
■ s_ramdata	0x20000000	0x20000000	128 KB
▼  CCMRAM	0x10000000		64 KB
>  .bss	0x10000b14		23,48 KB
>  .data	0x10000000	0x080128c0	2,77 KB
 ._user_heap_stack	0x10006900		1,5 KB

Dettaglio memoria progetto

CONFIGURAZIONE INTERRUPT

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Memory management fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Debug monitor	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pendable request for system service	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
System tick timer	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
PVD interrupt through EXTI line 16	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Flash global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
RCC global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM1 break interrupt and TIM9 global int...	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM1 update interrupt and TIM10 global i...	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM1 trigger and commutation interrupts ...	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM1 capture compare interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
TIM3 global interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
TIM4 global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
I2C2 event interrupt	<input checked="" type="checkbox"/>	14	0	<input checked="" type="checkbox"/>
I2C2 error interrupt	<input checked="" type="checkbox"/>	14	0	<input checked="" type="checkbox"/>
SPI2 global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>

Dettaglio interrupt

CONFIGURAZIONE PARTENZA DMA

Configuration

✓ DMA1

✓ DMA2

✓ MemToMem

DMA Request	Stream	Direction	Priority
TIM1_TRIG	DMA2 Stream 0	Memory To Peripheral	Very High

Add

Delete

DMA Request Settings

Peripheral		Memory
Mode	Normal	Increment Address <input type="checkbox"/>
Use Fifo <input checked="" type="checkbox"/>	Threshold Three Quarter...	Data Width Byte
	Burst Size	Single

Dettaglio configurazione DMA