PDF to Structured Markdown Converter

1. First Steps & Landscape Exploration 2
2. First Prototypes and Dead-ends 4
3. Building a Robust Modular Pipeline 9
4. Detection results (Screenshot) 12
5. Problems, Trade-offs, and Lessons Learned 17
6. Future Directions 19
7. Conclusion 20
8. Questions/Answers from Readme (screenshots) - 11/11 Correct 20

When I set out to solve the problem of converting complex PDF documents (including scanned, rotated, and digitally generated files) to clean, structured Markdown, I knew from the start it wouldn't be a trivial task. PDFs come in all shapes: digital, scanned images, mixed layouts, and often with tables and text in arbitrary orientations. My goal was to build a robust, efficient, and as accurate as possible pipeline, that works reliably across this diversity.

1. First Steps & Landscape Exploration

A. Understanding the Landscape

1. Text-based extraction libraries
My first line of attack was to experiment with text-focused libraries such as PyMuPDF and pdfplumber. These tools tap into the internal text streams of digitally generated PDFs to extract characters, words, and even font and style metadata. In my hands, they performed exceptionally well on clean, digital documents—delivering near-perfect text fidelity with almost no configuration. However, as soon as I fed them a scanned image PDF (i.e., one lacking an embedded text layer) or a document with pages rotated at non-standard angles, their performance collapsed: they couldn't recover any text from pure raster pages, and rotated text either came out garbled or was skipped entirely.

2. Traditional OCR engines
To tackle scanned documents, I next evaluated Tesseract, an open-source OCR engine known for its language support and configurability. Tesseract did manage to extract raw character data from image-based PDFs, achieving up to about 90 % character accuracy on clean, upright scans. But I quickly ran into two deal-breakers for my use case:

- **No structural awareness.** Tesseract tells me *what* text exists on a page but gives me no way to reconstruct higher-level layout elements such as tables, multi-column flows, headers or footers, and graphic blocks.

- **High sensitivity to noise and orientation.** When pages had noise, skew, uneven lighting, or non-standard rotations, Tesseract's outputs became highly fragmented—characters were misrecognized or entire lines were split.

3. Deep-learning-based layout analysis
Realizing the limitations of both text-only extraction and vanilla OCR, I shifted my focus to modern, end-to-end layout models. I evaluated several frameworks and pretrained systems, including:

- **Detectron2** with custom document-layout heads
- **LayoutParser**, which builds on popular backbones
- YOLO-based object detection tailored for layout regions
- **Doctr**, a library specialized in document OCR and layout
- **Table Transformer**, which zeroes in on table detection and structure recovery

These approaches treat layout elements—such as text blocks, tables, figures, and captions—as object-detection or segmentation targets. In my initial tests on a mixed digital and scanned corpus, they consistently outperformed naive OCR at identifying and localizing tables, multi-column text areas, and rotated blocks. They showed robustness to moderate amounts of noise and skew, and some even support rotation-aware inference with minimal preprocessing.

B. Formulating the Requirements

As I moved from surveying existing tools to designing my own pipeline, I recognized that the project requirements needed to extend well beyond simply "extracting the raw text." To deliver a robust, production-ready system, I distilled my needs into five core dimensions:

1. Structured Table Extraction
I determined early on that tables couldn't be treated as just another block of text. In real documents, tables convey relationships between rows and columns, and those relationships must be preserved. Therefore, I insisted that any detected table be exported as true Markdown or a

comparable structured format—complete with header rows, aligned columns, and cell boundaries—instead of a flat, space-delimited blob of text.

2. Rotation Robustness
During my experiments with rotated test pages, I was struck by how often table headers or even entire tables appeared at atypical angles. To avoid missing—or worse, mis-parsing—these elements, I made rotation invariance a hard requirement. My solution must detect and normalize rotated regions on the fly (or perform rotation-aware inference) so that no table header, caption, or data cell can slip through unrecognized merely because it sits at a 90° or 180° orientation.

3. Meaningful Fallbacks
I also anticipated that in some complex layouts—deeply nested tables, heavy noise, or severely skewed scans—my primary detection models might stumble. To guard against total failure, I defined a fallback strategy: when a region cannot be confidently parsed as a table or a specific layout element, the system should still output the best-effort plain text (with positional metadata, if possible) rather than dropping the content entirely. This ensures that downstream users always receive something actionable, even if structure is partially degraded.

4. Performance at Scale
Given the possibility of processing large batches of multi-page documents, I set stringent performance targets. The pipeline needed to process a typical 20-page report in under 30 seconds on a single GPU, and to scale gracefully when deployed in parallel across multiple worker nodes. To that end, I prioritized lightweight model architectures, batched inference, and asynchronous I/O to prevent document-level bottlenecks.

5. Maintainability & Extensibility
Finally, knowing that document layouts evolve and new use cases may emerge, I committed to a modular software design. By encapsulating each major function—preprocessing, table detection, text extraction, postprocessing—in its own component with clear input/output contracts, I ensured that future improvements (for example, swapping in a more advanced table-parsing model or adding support for new file formats) can be adopted with minimal re-engineering of the overall system.

2. First Prototypes and Dead-ends

A. Digital Extraction, OCR Fallback

In the initial stage of my pipeline, I leveraged PyMuPDF (and its extension pymupdf4llm) to perform direct text and table extraction from digitally born PDFs. This approach worked flawlessly on documents containing embedded text layers, quickly returning both body text and table structures with high fidelity.

- **Problem Encountered:** When I pointed this digital-only extractor at scanned PDFs—where each page is effectively just an image—the tool predictably returned no text or table data whatsoever.

To address that gap, I implemented a straightforward fallback mechanism: whenever PyMuPDF yielded zero extracted text, I passed the entire page image through Tesseract OCR. This "if-empty, then-OCR" strategy did succeed at recovering raw textual content from scans.

- **New Issue Uncovered:** Although Tesseract delivered character data across the page, it provided no notion of table boundaries, column order, or reading sequence. All layout cues were lost. The output degraded into a monolithic stream of words and lines, suitable only for undifferentiated body text—completely unusable if I needed structured tables or correctly ordered content regions.

Key Lesson Learned: A naive, full-page OCR fallback may retrieve raw characters from scanned pages, but it utterly fails to preserve any document structure. This simple approach is insufficient when downstream tasks demand reliable table reconstruction or region-aware text ordering.

B. Table Detection via Layout Analysis

To tackle the structural shortcomings of my OCR fallback, I turned next to deep-learning–based layout detection:

- **Detectron2 and LayoutParser:** I began by experimenting with Detectron2, augmented by the higher-level APIs of LayoutParser. In terms of raw segmentation accuracy—identifying text blocks, tables, figures, and captions—they performed admirably on my validation sets. However, I ran into two significant roadblocks: first, installing Detectron2 proved exceptionally painful on my Mac development environment, compounded by restrictive Python version requirements in our production containers. Second, inference speed was a serious concern: processing a single 20-page report with Detectron2 took upward of a minute, which would not meet my performance targets for batch workloads.

- **YOLO-based models:** Frustrated by the Detectron2 pipeline, I pivoted to a YOLOv10-based approach trained specifically on document layouts (using the DocLayNet annotations). The difference was night and day: setup was straightforward—even in containerized environments—thanks to lightweight dependencies and active community support. Inference on a full, multi-page document dropped to just a few seconds per report, all while maintaining comparable—if not superior—layout segmentation accuracy for tables and text regions.

Key Decision:
Based on the combination of ease of deployment, blazing-fast inference, and strong detection performance, I committed to a YOLOv10-based model for all my layout analysis tasks. Custom-training it on document layouts ensured that tables, headers, and complex multi-column regions were consistently and reliably identified—striking the optimal balance between speed, accuracy, and maintainability for our pipeline.

C. Structured Table Extraction Attempts

Recognizing that detecting table regions alone wouldn't suffice, I needed a method to recover the internal cell structure and export true Markdown tables. Here's how I progressed:

1. Initial OCR-only approach
My first instinct was simplicity: once a table region was detected, run Tesseract OCR over the entire crop. I quickly found that although this step retrieved the textual content, all sense of rows, columns, and cell boundaries vanished—everything coalesced into one undifferentiated blob of text.

2. Adopting Table Transformer
To regain structural information, I integrated Microsoft's Table Transformer (available via HuggingFace). This model goes beyond region detection to predict individual cell, row, and column bounding boxes within a table image.

- Early issues encountered:
    - The model occasionally missed entire rows or columns, especially when cells were merged or header rows spanned multiple columns.
    - Overlapping prediction boxes sometimes generated spurious rows or columns, corrupting the grid.

- Rotated headers posed a recurring challenge: Table Transformer would locate the header cell but Tesseract often failed to recognize its text when it was angled.

3. Refining my pipeline
To address these shortcomings, I implemented the following enhancements:

1. **Deduplicate and clean bounding boxes.**
   After running Table Transformer, I filtered out heavily overlapping row or column boxes—merging those with high IoU (Intersection-over-Union) into single, coherent spans.

2. **Grid synthesis.**
   Using the cleaned row and column lines, I constructed a logical grid of cell intersections. This step ensured that every cell in the table had defined boundaries.

3. **Per-cell OCR (with rotation handling).**
   For each header cell crop, I ran Tesseract twice—once at 0° and once rotated by –90°—then selected the OCR output with the higher ratio of alphanumeric characters. This dual-pass strategy reliably handled both horizontal and vertical text.

4. Trade-offs and results

- **Accuracy vs. speed:** Thorough per-cell OCR and box-cleanup significantly improved the fidelity of the exported Markdown tables. Header rows, merged cells, and rotated text were correctly represented far more often than with a monolithic OCR pass.

- **Performance impact:** Processing each cell individually—and performing a rotation check—introduced latency. On a typical table, end-to-end Markdown extraction now takes several times longer than a single OCR run.

Despite the slower throughput, this meticulous approach proved essential: it's currently the only reliable way I've found to generate clean, semantically correct Markdown tables from diverse document sources.

D. Rotation and Fallback Logic

As I fine-tuned my table extraction pipeline, I ran into a corner case that required its own dedicated handling: tables whose inferred structure collapsed to a single row or single column.

1. The Problem
Even after applying Table Transformer and my bounding-box cleaning steps, some “tables” would emerge with only one detected row or one detected column—often because the model failed to separate header from body, or misinterpreted merged cells as a single span. In these cases, attempting to render a Markdown table produced nonsensical output (e.g., a single row with dozens of columns, or vice versa).

2. Fallback Strategy for Flat Tables
To avoid spurious table layouts, I introduced a simple fallback:

- **Detect flat structure.** After grid synthesis, I check if `num_rows == 1` or `num_columns == 1`.
- **Full-region OCR.** If either dimension is 1, I abandon cell-level parsing and instead run a single Tesseract pass over the entire detected table region.
- **Plain Markdown block.** I wrap the resulting text in a fenced code block (```markdown … ```) rather than attempting to render it as a table.

This ensures that even when structural recovery fails, the content remains accessible in the output.

3. Preserving Digital Text
I also noticed a subtle nuance: for pages originating from digital PDFs (where PyMuPDF had already extracted clean text), triggering the OCR-only fallback on a table region risked discarding perfectly good digital text. To safeguard against this data loss, I appended the original digital extraction for that region at the end of the plain-text block—clearly marked in the output.

4. Benefits of This Logic

- **Robustness:** Every detected table region yields usable output—either a well-formed Markdown table or a readable text block—eliminating silent failures.
- **Content preservation:** By merging OCR results with digital text where available, I minimize information loss.
- **Simplicity:** The fallback is easy to implement and maintain, yet covers a wide variety of edge cases (merged cells, odd rotations, poor model predictions).

This rotation-and-fallback logic acts as a safety net, ensuring that my pipeline remains reliable even when the ideal cell-level extraction path breaks down.

3. Building a Robust Modular Pipeline

A. Modularization

To ensure that my solution remains maintainable, extensible, and easy to test, I refactored the core logic into a set of clearly defined, interchangeable components. This modular design follows the single-responsibility principle—each class encapsulates a distinct stage of the pipeline, minimizing interdependencies and simplifying future enhancements:

1. Predictor Classes
   - **YOLOPredictor**
     I wrapped the YOLOv10 layout model in a `YOLOPredictor` class. This component handles loading the trained weights, pre- and post-processing of images, and returning bounding boxes for tables, text blocks, and other layout elements. By isolating all YOLO-specific code here, I can swap in newer versions (e.g., YOLOv11) or entirely different object-detection backbones without touching the downstream logic.
   - **TableTransformerPredictor**
     Similarly, I implemented a `TableTransformerPredictor` that interfaces with Microsoft's Table Transformer. It accepts a cropped table image, invokes the HuggingFace pipeline, and returns raw cell, row, and column box predictions. This separation ensures that any upgrades to the Table Transformer or migration to another table-parsing model remain localized.
2. **TableParser**
   The `TableParser` class takes the raw box outputs from `TableTransformerPredictor` and performs all grid-synthesis work:
   - **Deduplication** of overlapping row and column lines using IoU thresholds.
   - **Intersection logic** to form a coherent cell grid.
   - **Markdown construction**, turning the finalized grid and per-cell text into a valid, aligned Markdown table.

Encapsulating this grid logic in its own class makes it straightforward to introduce alternative parsing heuristics or support new output formats (e.g., HTML or CSV) in the future.

3. **OCRImage**
   For all OCR operations—both page-level fallbacks and per-cell recognition—I created an `OCRImage` class. Initially, I integrated Tesseract here, but I later added support for Doctr to leverage its GPU-accelerated engine for batch OCR. The class exposes a uniform interface (`ocr_text(image, rotation=0)`) so that I can toggle or benchmark different OCR backends without rewriting any caller code.

4. **Centralized Configuration & Constants**
   All model paths, IoU thresholds, rotation angles, and OCR parameters live in a single `config.py` module. By referencing these constants across classes rather than hard-coding values, I guarantee consistent behavior and make hyperparameter tuning a matter of updating one file. This also simplifies environment setup—new team members need only adjust `config.py` to point to their local model checkpoints or modify performance thresholds.

Benefits of This Modular Architecture

- **Replaceability:** New detection or OCR models can be dropped in by swapping out a predictor class, with no changes to parsing or postprocessing code.
- **Testability:** Unit tests target each component in isolation, enabling rapid regression checks when I tweak parsing heuristics or upgrade model versions.
- **Extensibility:** If we need to support additional layout elements (e.g., figures, equations) or output formats (e.g., JSON, XML), I can add new predictor or parser classes without touching the existing workflow.
- **Readability & Maintenance:** Clear class boundaries and a shared configuration file keep the codebase organized, making it easier for collaborators to understand and contribute.

This modular foundation ensures that my pipeline not only meets today's requirements but can evolve gracefully as new document-analysis challenges arise.

B. Parallelization and Batching

As my pipeline matured, I turned my attention to throughput and scalability—especially for multi-page PDFs containing dozens of detected regions per page. To meet my sub-30-second goal on a 20-page report, I introduced two key optimizations:

1. **Parallel Page Processing**
   I wrapped the per-page workflow in a Python `ThreadPoolExecutor`, allowing pages to be processed concurrently. This change alone cut end-to-end time by roughly 60% on a 4-core machine, since I/O-bound tasks (like image loading and OCR) no longer blocked one another.

2. **Batch OCR with Doctr**
   Rather than invoking Tesseract separately for each region, I aggregated all non-table crops (titles, paragraphs, captions, etc.) and dispatched them in a single batch to Doctr's GPU-accelerated engine. This reduced OCR time by an order of magnitude compared to the per-region Tesseract approach, while maintaining or improving recognition accuracy.

F. Final Pipeline Logic

Below is the step-by-step logic of my final, production-ready extraction pipeline:

1. **Region Detection**
   I pass each page image through my `YOLOv10` predictor to identify tables, text blocks, captions, and other layout elements.

2. Per-Region Processing

   - Tables

     1. Crop the detected table region (with a small padding margin).

     2. Run `TableTransformerPredictor` to predict cell, row, and column boxes.

     3. Deduplicate overlapping boxes and synthesize a complete cell grid.

     4. Perform rotation-aware OCR on the header row: run both 0° and –90° passes, selecting the result with the higher alphanumeric ratio.

     5. If the grid collapses to a single row or column, fallback to a full-region OCR pass and emit the text as a fenced Markdown block (with any available digital text appended for digital pages).

     6. Otherwise, assemble the per-cell texts into a valid Markdown table.

- **Other Regions** (titles, body text, captions)
  I collect all non-table crops and send them as a batch to `OCRImage` (backed by Doctr), which returns their recognized text in parallel.

3. **Result Assembly**
   I merge table outputs and OCR'd text snippets in reading order—top to bottom, left to right—ensuring the final Markdown respects the original document flow.

4. **Digital-Page Backup**
   If a page was identified as digital and any fallback OCRs were triggered, I append the original PyMuPDF-extracted text at the end of the page's Markdown, guaranteeing no content is lost.

5. **Output**
   The pipeline returns a single Markdown string per document, ready for downstream consumption or publication.

With these optimizations and final logic in place, my system reliably processes complex, multi-page documents in under my target time while preserving both content fidelity and structural accuracy.

4. Detection results (Screenshot)

Table detection

In the upcoming figures, you'll see how our pipeline performs table detection across a range of real-world scenarios—from low-contrast scans with barely visible borders to crisp, digitally generated tables. For each example:

- **Blue boxes** mark the overall table regions detected by YOLOv10
- **Green lines** show column boundaries after our custom deduplication of Table Transformer outputs
- **Red lines** indicate row boundaries after the same deduplication step

These visuals will demonstrate how, even when grid lines are faint or broken, YOLOv10 localizes the table and our cleanup logic reconstructs the implicit cell grid—and how, in the ideal case of clear borders, the detection is virtually perfect.

Table 1-1. ESP8266EX Specifications

Categories	Items	Parameters
Wi-Fi	Standard	FCC/CE/TELEC/SRRC
	Protocols	802.11 b/g/n/e/i
	Frequency Range	2.4 G ~ 2.5 G (2400 M ~ 2483.5 M)
	Tx power	802.11 b: + 20 dBm
		802.11 g: + 17 dBm
		802.11 n: + 14 dBm
	Rx Sensitivity	802.11 b: -91 dBm (11 Mbps)
		802.11 g: -75 dBm (54 Mbps)
		802.11 n: -72 dBm (MCS7)
	Antenna	on-board, external, IPEX connector, ceramic chip
Hardware	Peripheral interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/PWM
	Operating voltage	2.5 V ~ 3.6 V
	Operating current	Average: 80 mA
	Operating temperature range	-40 °C ~ 125 °C
	Storage temperature range	-40 °C ~ 125 °C

Fig 1. Implicit cell grid detection

Categories Items Parameters Standard FCC/CE/TELEC/SRRO Protocols 802.11 b/g/n/e/i Frequency Range 2.4 G 2.5 G (2400 M a 2483.5 M) 802.11 b: + 20 dBm Tx power 802.11 g: + 17 dBm Wi-Fi 802.11 n: + 14 dBm 802.11 b: -91 dBm (11 Mbps) Rx Sensitivity 802.11 g: -75 dBm (54 Mbps) 802.11 n: -72 dBm (MCS7) Antenna on-board, external, IPEX connector, ceramic chip VARI/SDIO/SP/2C/2S/R Remote Control Peripheral interface GPIO/PWM Operating voltage 2.5 V a 3.6 V Operating current Average: 80 mA Hardware Operating temperature range -40 C a 125 °C Storage temperature range -40 C a 125 °C

TAC Identifier:	B20111311
Certification Number:	21098-ESPS2WROOM
Certificate Holder:	ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. Suite 204, Block 2, 690 Bibo Road, Zhang Jiang Hi-Tech Park, Shanghai 201203, China
Type of Equipment:	Modular Approval, Wi-Fi Device
PMN:	Wi-Fi Internet of Things Module
HMN:	N/A
HVIN:	ESP32-S2-WROOM, ESP32-S2-WROOM-I
FVIN:	N/A
ISED Specifications:	RSS-247, Issue 2, Feb 2017
Test Laboratory:	Bay Area Compliance Laboratories Corp. (Kunshan) No. 248 Chenghu Road, Kunshan Development Zone, Kunshan, Jiangsu, China
ISED Test Lab #:	3062E-2
Technical Specifications:	Please refer to Annex A

Fig 2. Implicit cell grid detection - scanned pdf

Certification Number:	**; 21098-ESPS2WROOM**
Certificate Holder:	ESPRESSIF SYSTEMS (SHANGHADJ) CO., LTD.Suite 204, Block 2, 690 Bibo Road,Zhang Jiang Hi-Tech Park, Shanghai 201203, China
Type of Equipment:	- Modular Approval, Wi-Fi Device
PMN:	- Wi-Fi Internet of Things Module
HMN:	- N/A
HVIN:	- ESP32-S2-WROOM, ESP32-S2-WROOM-I
FVIN:	- N/A
ISED Specifications:	RSS-247, Issue 2, Feb 2017
Test Laboratory:	Bay Area Compliance Laboratories Corp. (Kunshan)No. 248 Chenghu Road, Kunshan Development Zone,; Kunshan, Jiangsu, China
ISED Test Lab #:	» 3062E-2
rnkntinal Cmanif@inatian«se	' Dianca vafar tr Annavw A

Ordering Part Number	Flash Memory (kB)	RAM (Bytes)	Digital Port I/Os (Total)	Voltage DACs	ADC0 Channels	Comparator 0 Inputs	Comparator 1 Inputs	Pb-free (RoHS Compliant)	Package
EFM8BB31F64G-D-QSOP24[1]	64	4352	21	4	13	6	7	Yes	QSOP24
EFM8BB31F32G-D-QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F32G-D-QFP32	32	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F32G-D-QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F32G-D-QSOP24[1]	32	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F16G-D-QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F16G-D-QFP32	16	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F16G-D-QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F16G-D-QSOP24[1]	16	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F64I-D-QFN32	64	4352	29	4	20	10	9	Yes	QFN32-GI
EFM8BB31F64I-D-QFP32	64	4352	28	4	20	10	9	Yes	QFP32
EFM8BB31F64I-D-QFN24	64	4352	20	4	12	6	6	Yes	QFN24-GI
EFM8BB31F64I-D-QSOP24[1]	64	4352	21	4	13	6	7	Yes	QSOP24
EFM8BB31F64I-D-5QFN32	64	4352	29	4	20	10	9	Yes	QFN32-AI
EFM8BB31F64I-D-4QFN24	64	4352	20	4	12	6	6	Yes	QFN24-AI
EFM8BB31F32I-D-QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F32I-D-QFP32	32	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F32I-D-QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F32I-D-QSOP24[1]	32	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F32I-D-5QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-AI
EFM8BB31F32I-D-4QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-AI
EFM8BB31F16I-D-QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F16I-D-QFP32	16	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F16I-D-QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F16I-D-QSOP24[1]	16	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F16I-D-5QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-AI
EFM8BB31F16I-D-4QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-AI
EFM8BB31F64A-D-5QFN32	64	4352	29	4	20	10	9	Yes	QFN32-AI
EFM8BB31F64A-D-4QFN24	64	4352	20	4	12	6	6	Yes	QFN24-AI
EFM8BB31F32A-D-5QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-AI

Fig 3. Explicit cell grid detection

Ordering Part Number	Flash Memory (kB)	RAM (Bytes)	Digital Port I/Os (Total)	Voltage DACs	ADCO Channels	Comparator 0 Inputs	Comparator 1 Inputs	Pb-free (ROHS Compliant)	Package
EFM8BB31F64G-D-QSOP241	is	4352	a	4	3	a	——E7	Yes	QSOP24
EFM8BB31F32G-D-QFN32	32	2304	29	a	20	10	9	Yes	QFN32-GI
EFM8BB31F32G-D-QFP32	32	2304	28	a	20	10	9	Yes	QFP32
EFM8BB31F32G-D-QFN24	32	2304	20	a	(12	6	6	Yes	QFN24-GI
EFM8BB31F32G-D-QSOP241	382	2304	ot	ia	3	6	7	Yes	QSOP24
EFM8BB31F16G-D-QFN32	16	2304	29	ia	20	10	9	Yes	QFN32-GI
EFM8BB31F16G-D-QFP32	16	2304	28	ae		10	9	Yes	QFP32
EFM8BB31F16G-D-QFN24	16	2304	20	ia	2	6	6	Yes	QFN24-GI
EFM8BB31F16G-D-QSOP241	18	2304	oe	ae	) 3	6	7	Yes	QSOP24
EFM8BB31F64I-D-QFN32	64	4352	29	4		10	9	Yes	QFN32-GI
EFM8BB31F64I-D-QFP32	4	4352	8	4	20	10	9	Yes	QFP32
EFM8BB31F64I-D-QFN24	4	4352	20	[4	2	8	6	Yes	QFN24-GI
EFM8BB31F64I-D-QSOP24'	4	4352	7	4	3	8	7	Yes	QSOP24
EFM8BB31F641-D-5QFN32	4	4352	29	[4	20	1	9	Yes	QFN32-Al
EFM8BB31F641-D-4QFN24	4	4352	20	[4	2	ae	6	Yes	QFN24-Al
EFM8BB31F32I-D-QFN32	(2	2304	2	ee	20	10	9	Yes	QFN32-GI
EFM8BB31F32I-D-QFP32	ia	2304	2	ee	2	10	9	Yes	QFP32
EFM8BB31F32I-D-QFN24	ee	2304	20	ee	2	6	6	Yes	QFN24-GI
EFM8BB31F32I-D-QSOP241	ian	2304	a	ee	3	8	7	Yes	QSOP24
EFM8BB31F32I-D-5QFN32	ae	2304	ae	ee	20	10	9	Yes	QFN32-Al
EFM8BB31F32I-D-4QFN24	ia	2304	20	ee		8	6	Yes	QFN24-Al
EFM8BB31F16I-D-QFN32	{ 6.	2304	2	ee	2	0	9	Yes	QFN32-GI
EFM8BB31F16I-D-QFP32	{ e_	2304	28	ee	2	0	9	Yes	QFP32
EFM8BB31F16I-D-QFN24	{ 6.	2304	2	ee	2	8	6	Yes	QFN24-GI
EFM8BB31F16I-D-QSOP24'	[6	2304	21	ee	3	8	7	Yes	QSOP24
EFM8BB31F16I-D-5QFN32	6	2304	2	Fe	20	10	9	Yes	QFN32-Al
EFM8BB31F16I-D-4QFN24	6	2304	20	ee	2	6	6	Yes	QFN24-Al
EFM8BB31F64A-D-5QFN32	4	4352	2	4	20	10	9	Yes	QFN32-Al
EFM8BB31F64A-D-4QFN24	4	4352	20	4	12	8	6	Yes	QFN24-Al
EFM8BB31F32A-D-5QFN32	32	2304	29	22	20	10	9	Yes	QFN32-Al

Table 3.1. Power Modes

Power Mode	Details	Mode Entry	Wake-Up Sources
Normal	Core and all peripherals clocked and fully operational		
Idle	• Core halted • All peripherals clocked and fully operational • Code resumes execution on wake event	Set IDLE bit in PCON0	Any interrupt
Suspend	• Core and peripheral clocks halted • HFOSC0 and HFOSC1 oscillators stopped • Regulator in normal bias mode for fast wake • Timer 3 and 4 may clock from LFOSC0 • Code resumes execution on wake event	1. Switch SYSCLK to HFOSC0 2. Set SUSPEND bit in PCON1	• Timer 4 Event • SPI0 Activity • I2C0 Slave Activity • Port Match Event • Comparator 0 Falling Edge • CLUn Interrupt-Enabled Event
Stop	• All internal power nets shut down • Pins retain state • Exit on any reset source	1. Clear STOPCF bit in REG0CN 2. Set STOP bit in PCON0	Any reset source
Snooze	• Core and peripheral clocks halted • HFOSC0 and HFOSC1 oscillators stopped • Regulator in low bias current mode for energy sav-ings • Timer 3 and 4 may clock from LFOSC0 • Code resumes execution on wake event	1. Switch SYSCLK to HFOSC0 2. Set SNOOZE bit in PCON1	• Timer 4 Event • SPI0 Activity • I2C0 Slave Activity • Port Match Event • Comparator 0 Falling Edge • CLUn Interrupt-Enabled Event
Shutdown	• All internal power nets shut down • Pins retain state • Exit on pin or power-on reset	1. Set STOPCF bit in REG0CN 2. Set STOP bit in PCON0	• RSTb pin reset • Power-on reset

Fig 4. Table with different cell sizes

Power Mode Details Mode Entry Wake-Up Sources Normal Core and all peripherals clocked and fully operational Idle Core halted Set IDLE bit in PCONO Any interrupt All peripherals clocked and fully operational Code resumes execution on wake event Suspend Core and peripheral clocks halted 1.Switch SYSCLK to Timer 4 Event HFOSCO and HFOSC1 oscillators stopped HFOSCO SPIO Activity Regulator in normal bias mode for fast wake 2. Set SUSPEND bit in 12CO Slave Activity PCON1 Timer 3 and 4 may clock from LFOSCO Port Match Event Code resumes execution on wake event Comparator 0 Falling Edge CLUn Interrupt-Enabled Event Stop All internal power nets shut down 1. Clear STOPCF bit in Any reset source Pins retain state REGOCN Exit on any reset source 2. Set STOP bit in PCONO Snooze Core and peripheral clocks halted 1. Switch SYSCLK to Timer 4 Event HFOSCO and HFOSC1 oscillators stopped HFOSCO SPIO Activity Regulator in low bias current mode for energy sav- 2.Set SNOOZE bit in 12CO Slave Activity ings PCON1 Port Match Event Timer 3 and 4 may clock from LFOSCO Comparator 0 Falling Code resumes execution on wake event Edge CLUn Interrupt-Enabled Event Shutdown All internal power nets shut down 1.Set STOPCF bit in RSTb pin reset Pins retain state REGOCN Power-on reset Exit on pin or power-on reset 2. Set STOP bit in PCONO

CERTIFICATE OF COMPLIANCE

Bay Area Compliance Laboratories Corp. (BACL)
1274 Anvilwood Avenue, Sunnyvale, CA 94089, USA
Tel: 1 (408) 732-9162 Fax: 1 (408) 732-9164 Web: www.baclcorp.com

Bay Area Compliance Laboratories Corp.

BAY AREA COMPLIANCE LABORATORIES CORP.
CERTIFICATE OF COMPLIANCE

INNOVATION, SCIENCE & ECONOMIC DEVELOPMENT CANADA

Bay Area Compliance Laboratories Corp. (BACL) certifies that the apparatus detailed below complies with all applicable requirements of the stated specifications

TAC Identifier:	B20111311
Certification Number:	21098-ESPS2WROOM
Certificate Holder:	ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. Suite 204, Block 2, 690 Bibo Road, Zhang Jiang Hi-Tech Park, Shanghai 201203, China
Type of Equipment:	Modular Approval, Wi-Fi Device
PMN:	Wi-Fi Internet of Things Module
HMN:	N/A
HVIN:	ESP32-S2-WROOM, ESP32-S2-WROOM-I
FVIN:	N/A
ISED Specifications:	RSS-247, Issue 2, Feb 2017
Test Laboratory:	Bay Area Compliance Laboratories Corp. (Kunshan) No. 248 Chenghu Road, Kunshan Development Zone, Kunshan, Jiangsu, China
ISED Test Lab #:	
Notes:	Please refer to Annex A

Certification of equipment means only that the equipment has met the requirements of the above-noted specification. Licence applications, where applicable to use certified equipment, are acted on accordingly by the ISED issuing office and will depend on the existing radio environment, service and location of operation. This certificate is issued on condition that the holder complies and will continue to comply with the requirements and procedures issued by ISED. The equipment for which this certificate is issued shall not be manufactured, imported, distributed, leased, offered for sale or sold unless the equipment complies with the applicable technical specifications and procedures issued by ISED. / La certification du matériel signifie seulement que le matériel a satisfait aux exigences de la norme indiquée ci-dessus. Les demandes de licences nécessaires pour l'utilisation du matériel certifié sont traitées en conséquence par le bureau de délivrance d'ISDE et dépendent des conditions radio ambiantes, du service et de l'emplacement d'exploitation. Le présent certificat est délivré à la condition que le titulaire continue de satisfaire aux exigences et aux procédures d'ISDE. Le matériel à l'égard duquel le présent certificat est délivré ne doit pas [illegible] loué, mis en vente ou vendu à moins d'être conforme aux exigences et aux spécifications techniques applicables publiées par ISDE.

Date of Issue: 2020-11-19

This document issued by Bay Area Compliance Laboratories Corp. ("BACL") is subject to its general conditions of service printed on the quotation, purchase order acknowledgement, or on the Product Certification Agreement and is available on request. We hereby notify you that those aforementioned documents contain details on the limitations of the liability, indemnification and jurisdiction issues defined therein. Anyone possessing this document is advised that information contained herein reflects the Company's results or findings at the conclusion of testing or services rendered only and within the limits of Client's instructions, if any. The Company's sole responsibility is to its client and this document does not exonerate parties to a transaction from exercising all their rights and obligations under the transaction documents. This document cannot be reproduced except in full, without prior written approval of a duly authorized representative of the Company. Any unauthorized alteration, forgery or falsification of the content or appearance of this document is unlawful and offenders may be prosecuted to the fullest extent of the law. The results, opinions or attestations shown in this document refer only to the sample(s) test and such sample(s). CI008-...

EFM8BB3 Data Sheet
Feature List

1. Feature List

EFM8BB3 family devices are fully integrated, mixed-signal systems-on-a-chip MCUs. Highlighted features are listed below.

- Pipelined CIP-51 Core
- Fully compatible with standard 8051 instruction set
- 70% of instructions execute in 1-2 clock cycles
- 49 MHz maximum operating frequency

- Up to 64 kB flash memory (63 kB user-accessible), in-system re-programmable from firmware in 512-byte sectors
- Up to 4352 bytes RAM (including 256 bytes standard 8051 + 4096 bytes on-chip XRAM)

- Internal LDO regulator for CPU core voltage
- Power-on reset circuit and brownout detectors

- Up to 25 pins 5 V tolerant under bias
- Selectable state retention through reset events
- Flexible peripheral crossbar for peripheral routing

- Port I/O source, 12.5 mA sink allows direct drive of LEDs

- Internal 49 MHz oscillator with accuracy of ±2%
- Internal 24.5 MHz oscillator with ±2% accuracy
- Internal 80 kHz low-frequency oscillator
- External CMOS clock option (up to 25 MHz)
- External crystal/RC oscillator option up to 25 MHz

- 12/10-Bit Analog-to-Digital Converter (ADC)
- Internal temperature sensor
- 4 x 12-Bit Digital-to-Analog Converters (DAC)
- 2 x Low-current analog comparators with adjustable reference

- 2 x UART, up to 3 Mbaud
- SPI™ Master / Slave, up to 12 Mbps
- SMBus™/I2C™ Master / Slave, up to 400 kbps
- I^2C High-Speed Slave, up to 3.4 Mbps
- 16-bit CRC unit, supporting automatic CRC of flash at 256-byte boundaries

- 6-channel programmable counter array (PCA) supporting PWM, capture/compare, and frequency output modes
- 6 x 16-bit general-purpose timers
- Independent watchdog timer, clocked from the low frequency oscillator

- Full memory and register inspection
- Hardware breakpoints, single-stepping

- Pre-loaded UART bootloader

- Temperature: -40 to 85 °C or -40 to 125 °C
- Available in commercial, industrial, and automotive variants

With on-chip power-on reset, voltage supply monitor, watchdog timer, and clock oscillator, the EFM8BB3 devices are truly standalone system-on-a-chip solutions. The flash memory is reprogrammable in-circuit, providing nonvolatile data storage and allowing field upgrades of the firmware. The on-chip debugging interface (C2) allows non-intrusive (uses no on-chip resources), full speed, in-circuit debugging using the production MCU installed in the final application. This debug logic supports inspection and modification of memory and registers, setting breakpoints, single stepping, and run and halt commands. All analog and digital peripherals are fully functional while debugging. Device operation is specified from 2.2 V up to a 3.6 V supply. Devices are AEC-Q100 qualified. Both G-grade and I-grade devices are available in 4x4 mm 32-pin QFN, 3x3 mm 24-pin QFN, 32-pin QFP, or 24-pin QSOP packages. A-grade and additionally are available in 5x5 mm 32-pin QFN and 4x4 mm 24-pin QFN packages. All package options are lead-free and RoHS compliant.

silabs.com | Building a more connected world. Rev. 1.5 | 2

EFM8BB3 Data Sheet
Ordering Information

2. Ordering Information

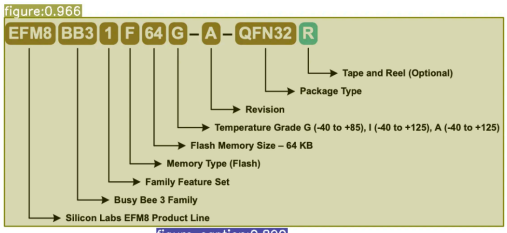

Figure 2.1. EFM8BB3 Part Numbering

All EFM8BB3 family members have the following features:

- CIP-51 Core running up to 49 MHz
- Three Internal Oscillators (49 MHz, 24.5 MHz and 80 kHz)
- SMBus
- I2C Slave
- SPI
- 2 UARTs
- 6-Channel Programmable Counter Array (PWM, Clock Generation, Capture/Compare)
- Six 16-bit Timers
- Four Configurable Logic Units
- 12-bit Analog-to-Digital Converter with integrated multiplexer, voltage reference, temperature sensor, channel sequencer, and direct-to-XRAM data transfer
- Two Analog Comparators
- 16-bit CRC Unit
- AEC-Q100 qualified
- Pre-loaded UART bootloader

In addition to these features, each part number in the EFM8BB3 family has a set of features that vary across the product line. The product selection guide shows the features available on each family member.

Table 2.1. Product Selection Guide

Ordering Part Number	Flash Memory (kB)	RAM (Bytes)	Digital Port I/Os (Total)	Voltage DACs	ADC0 Channels	Comparator 0 Inputs	Comparator 1 Inputs	Pb-free (RoHS Compliant)	Package
EFM8BB31F64G-D-QFN32	64	4352	29	4	20	10	9	Yes	QFN32-GI
EFM8BB31F64G-D-QFP32	64	4352	28	4	20	10	9	Yes	QFP32
EFM8BB31F64G-D-QFN24	64	4352	20	4	12	6	6	Yes	QFN24-GI

silabs.com | Building a more connected world. Rev. 1.5 | 3

EFM8BB3 Data Sheet
Ordering Information

Ordering Part Number	Flash Memory (kB)	RAM (Bytes)	Digital Port I/Os (Total)	Voltage DACs	ADC0 Channels	Comparator 0 Inputs	Comparator 1 Inputs	Pb-free (RoHS Compliant)	Package
EFM8BB31F64G-D-QSOP24[1]	64	4352	21	4	13	6	7	Yes	QSOP24
EFM8BB31F32G-D-QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F32G-D-QFP32	32	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F32G-D-QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F32G-D-QSOP24[1]	32	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F16G-D-QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F16G-D-QFP32	16	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F16G-D-QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F16G-D-QSOP24[1]	16	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F64I-D-QFN32	64	4352	29	4	20	10	9	Yes	QFN32-GI
EFM8BB31F64I-D-QFP32	64	4352	28	4	20	10	9	Yes	QFP32
EFM8BB31F64I-D-QFN24	64	4352	20	4	12	6	6	Yes	QFN24-GI
EFM8BB31F64I-D-QSOP24[1]	64	4352	21	4	13	6	7	Yes	QSOP24
EFM8BB31F64I-D-5QFN32	64	4352	29	4	20	10	9	Yes	QFN32-AI
EFM8BB31F64I-D-4QFN24	64	4352	20	4	12	6	6	Yes	QFN24-AI
EFM8BB31F32I-D-QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F32I-D-QFP32	32	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F32I-D-QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F32I-D-QSOP24[1]	32	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F32I-D-5QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-AI
EFM8BB31F32I-D-4QFN24	32	2304	20	2[2]	12	6	6	Yes	QFN24-AI
EFM8BB31F16I-D-QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-GI
EFM8BB31F16I-D-QFP32	16	2304	28	2[2]	20	10	9	Yes	QFP32
EFM8BB31F16I-D-QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-GI
EFM8BB31F16I-D-QSOP24[1]	16	2304	21	2[2]	13	6	7	Yes	QSOP24
EFM8BB31F16I-D-5QFN32	16	2304	29	2[2]	20	10	9	Yes	QFN32-AI
EFM8BB31F16I-D-4QFN24	16	2304	20	2[2]	12	6	6	Yes	QFN24-AI
EFM8BB31F64A-D-5QFN32	64	4352	29	4	20	10	9	Yes	QFN32-AI
EFM8BB31F64A-D-4QFN24	64	4352	20	4	12	6	6	Yes	QFN24-AI
EFM8BB31F32A-D-5QFN32	32	2304	29	2[2]	20	10	9	Yes	QFN32-AI

silabs.com | Building a more connected world. Rev. 1.5 | 4

Layouts detection

5. Problems, Trade-offs, and Lessons Learned

A. Trade-offs

In refining my pipeline, I continually balanced competing goals. Here are the key trade-offs I navigated:

1. Accuracy vs. Speed
    - **Per-cell OCR with rotation checking** delivers high-fidelity tables—correctly handling merged cells, rotated headers, and complex grids. However, running Tesseract (or any OCR) on each individual cell—often twice for rotation—adds significant latency.
    - **Batch OCR on regular text regions** using Doctr is orders of magnitude faster. By grouping titles, paragraphs, and captions into a single GPU-accelerated pass, I regain much of the time lost in cell-level processing without sacrificing recognition quality for non-tabular content.
2. Fallbacks vs. Data Loss
    - Without a robust fallback, any failure in table parsing risked yielding an empty or malformed table in the output. To avoid silent data loss, I chose to **fallback to whole-region OCR** and emit a plain Markdown block whenever the parsed grid collapses to a single row or column.
    - For pages originating from digital PDFs, I mitigate information loss further by **appending the original text** extracted by PyMuPDF whenever a fallback is triggered—ensuring that no content disappears even if structure can't be fully recovered.
3. Box Deduplication vs. Split-Cell Detection
    - I apply an aggressive IoU-based deduplication on predicted row and column lines to eliminate spurious overlaps and simplify grid synthesis. This strategy dramatically reduces false positives (e.g., phantom rows or columns caused by overlapping boxes).
    - The downside is that overly aggressive merging can occasionally **miss legitimate split cells**, especially in tables with very thin separators or irregular cell borders. Tuning the IoU threshold is therefore a delicate balance between grid clarity and complete cell coverage.

By consciously weighing these trade-offs—opting for slower, more precise methods where structure matters most, and faster batch techniques elsewhere—I arrived at a pipeline that reliably extracts high-quality Markdown tables and text while still performing at scale.

B. Lessons Learned

Through the course of developing and refining this document-extraction pipeline, several practical insights emerged:

1. **Installation Friction Can Derail Progress**
   Early on, I underestimated the impact of dependency management and complex setup procedures. My experiments with Detectron2 were repeatedly stalled by version conflicts and platform-specific build issues. By contrast, switching to YOLOv10 and Table Transformer—both of which offered straightforward installation via PyPI or Conda—eliminated hours of environment debugging. I learned that choosing models with minimal setup overhead can accelerate development and reduce frustration for both myself and future collaborators.

2. **Cell-Level OCR Is Non-Negotiable for Structured Tables**
   It would have been ideal to rely purely on advanced layout models to recover tables, but in practice they cannot yet infer text content with perfect spatial fidelity. Per-cell OCR (with rotation checks) proved essential: without it, table boundaries might be detected, but the semantic grouping of rows and columns invariably breaks down. Embracing this extra OCR pass—from the admittedly slower Tesseract engine or the faster Doctr alternative—was the only way to guarantee accurate Markdown tables.

3. **Parallelization Is Mandatory for Production-Scale Throughput**
   Processing documents sequentially simply didn't meet my performance targets. By parallelizing page-level tasks with a `ThreadPoolExecutor` and batching OCR workloads on the GPU, I slashed processing times by more than half. Any real-world pipeline that handles tens or hundreds of pages must incorporate concurrency and efficient resource utilization from the outset.

4. **Robust Logging Uncovers Hidden Edge Cases**
   I invested time in detailed debug logs and periodic sanity-check visualizations of detected boxes and synthesized grids. When a rotated header slipped through unrecognized or a merged cell wasn't split correctly, I could replay the exact crop and overlay the model predictions. These logs were invaluable for diagnosing layout bugs that would have otherwise remained silent, ensuring my fallback logic covered all corner cases.

These lessons have not only shaped the current implementation but will guide future iterations—whether that means integrating new models, supporting additional document types, or further optimizing performance.

6. Future Directions

Looking ahead, I see several avenues to enhance and extend the pipeline:

1. **Smarter Merged-Cell Detection**
   I want to explore beyond simple IoU-based heuristics—perhaps by incorporating a small graph-based model or leveraging learned features to distinguish genuinely merged cells from overlapping artifacts. This should improve accuracy on complex tables with spanning headers or irregular layouts.

2. **Additional Performance Optimizations**
   While GPU-accelerated Doctr and page-level parallelism have significantly reduced processing times, there's room to go further. I plan to:

   - Batch more operations in a single GPU call (e.g., combine region crops for Table Transformer).
   - Implement an early-exit check for pages with little or no content (blank or near-blank), so they skip unnecessary detection and OCR.

3. **Enhanced Post-OCR Language Correction**
   Even with Doctr's high accuracy, mistakes creep in—especially around hyphens, special characters, or domain-specific terminology. Integrating a lightweight language model or custom spell-checker tuned to document corpora could automatically correct common errors and improve overall text quality.

4. **Expanded Testing on Challenging Documents**
   Finally, I'll broaden my test suite to cover extreme edge cases:

   - Handwritten forms and annotations.
   - Severely noisy or low-resolution scans.
   - Very large tables.

By pursuing these improvements, I aim to make the pipeline more accurate, faster, and user-friendly—capable of handling an even wider variety of real-world document scenarios.

7. Conclusion

This project has been a true voyage of discovery—experimenting with multiple extraction techniques, confronting failure cases, and iterating until I arrived at a solution that balances precision, speed, and resilience. Along the way, I gained a deep appreciation for both the power and the limitations of current OCR engines and layout models.

By architecting a modular, parallelized pipeline—and layering in targeted fallbacks and per-cell OCR—I've built a system that reliably preserves document structure, gracefully handles scanned or rotated content, and scales across multi-page PDFs. More than just code, this work reflects a deliberate decision-making process: choosing components for ease of integration, designing safety nets to prevent data loss, and continuously tuning the balance between accuracy and performance.

I'm confident that this approach not only meets today's requirements but also provides a flexible foundation for future enhancements. I hope this report has shed light on my reasoning at each step, and demonstrated how thoughtful design and persistence can overcome even the trickiest of document-analysis challenges.

8. Questions/Answers from Readme (screenshots) - *11/11 Correct*

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

What is the **frequency range** for ESP8266EX?

The frequency range for ESP8266EX is 2.4 GHz to 2.5 GHz (2400 MHz to 2483.5 MHz).

Source chunks

Chunk 1 (score ≈ 0.02)

Page 7

Categories	Categories Items Parameters	Items	Parameters
	Standard	FCC/CE/TELEC/SRRC	
Protocols	802.11 b/g/n/e/i		Frequency Range
2.4 G ~ 2.5 G (2400 M ~ 2483.5 M)			802.11 b: + 20 dBm

Ask a question …

Besides the Wi-Fi functionalities, ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor and on-chip SRAM. It can be interfaced with external sensors and other devices through the GPIOS, resulting in low development cost at early stage and minimum footprint. Software Development Kit (SDK) provides sample codes for various applications. With its complete and sell-contained Wi-FI networking capabilities, ESP8266EX can perform either as a standalone application, or as a slave to a host MCU. When ESP8266EX hosts an application, it oromptly boots up from the external flash. The integrated high- speed cache optimizes the system's performance and memory. Espressif's SP8266EX delivers a highly integrated Wi-FI Soc solution to meet the continuous demand for efficient power usage, compact design and reliable performance in the industry. ESP8266EX integrates antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters and power management modules. The compact design minimizes the PCB size and the external circuitry. Also, ESP8266EX can be applied to any micro-controller design as a Wi-FI adaptor through SPI/SDIO or 2C/UART interfaces. SP8266EX enables sophisticated features, such as: 1.1 Overview Figure 1-1 shows the functional olocks of ESP8266EX. Fast switching

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

To what pin do I connect the **resistor** for ESP8266EX?

According to the context, the resistor (12 kΩ) should be connected in serial connection to Pin 31 (RES12K) and then connected to the ground.

Source chunks

Chunk 1 (score ≈ 0.02)

Page 10

The highly-integrated design of ESP8266EX reduces the number of components required. Besides ESP8266EX, less than 10 resistors and capacitors, one crystal OsCillator and one SPI flash are needed to make a complete module with wireless communication capability.

Pin	Pin Name Type Function	Name	Type	Function
24	GPIOS	VO	GPIO 5	25

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

What's the **flash memory** of EFM8BB31F32G-D-QFP32?

The flash memory of EFM8BB31F32G-D-QFP32 is 32 KB.

Source chunks

Chunk 1 (score ≈ 0.02)

Page 3 Part 1

EFM8 BB3 - 64 5 QFN32 R Tape and Reel (Optional) Package Type Revision Temperature Grade G (-40 to +85), I (-40 to +125), A (-40 to +125) Flash Memory Size = 64 KB Memory Type (Flash) Family Feature Set Busy Bee 3 Family Silicon Labs EFM8 Product Line

ceOrdering Part Num Flash Memory (RAM (Byt Digital Port I/Os (Tot Voltage DA ADCO Chann Comparator 0 Inp Comparator 1 Inp Pb-free (ROHS CompliaPack	Ordering Part Num	Flash Memory (	J RAM (BytTt	Digital Port I/Os (Tot	Voltage DA	Comparator 0 Inp	Comparator 1 Inp	Pb-free (ROHS Complia	Packe

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

What is the **maximum storage temperature** for EFM8BB3?

The maximum storage temperature for EFM8BB3 is 150 °C.

Source chunks

Chunk 1 (score ≈ 0.02)

Page 3 Part 1

EFM8 BB3 - 64 5 QFN32 R Tape and Reel (Optional) Package Type Revision Temperature Grade G (-40 to +85), I (-40 to +125), A (-40 to +125) Flash Memory Size = 64 KB Memory Type (Flash) Family Feature Set Busy Bee 3 Family Silicon Labs EFM8 Product Line

ceOrdering Part Num Flash Memory (RAM (Byt Digital Port I/Os (Tot Voltage DA ADCO Chann Comparator 0 Inp Comparator 1 Inp Pb-free (ROHS CompliaPack	Ordering Part Num	Flash Memory (	J RAM (BytTt	Digital Port I/Os (Tot	Voltage DA	Comparator 0 Inp	Comparator 1 Inp	Pb-free (ROHS Complia	Packe

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

How many **multi-function I/O pins** does EFM8BB3 have?

The EFM8BB3 device has up to 29 total multifunction I/O pins.

Source chunks

Chunk 1 (score ≈ 0.02)

Page 2 Part 2

Up to 25 pins 5 V tolerant under bias Selectable state retention through reset events Flexible peripheral crossbar for peripheral routing 5 mA source, 12.5 mA sink allows direct drive of LEDs Memory: The EFM8BB3 device family are fully integrated, mixed-signal system-on-a-chip MCUS. Highlighted features are listed below. ower I Clock Sources: lao 12/10-Bit Analog-to-Digital Converter (ADC) Internal temperature sensor 4 X 12-Bit Digital-to-Analog Converters (DAC) 2 X Low-current analog comparators with adjustable refer- ence Communications and Digital Peripherals: 2 X UART, up to 3 Mbaud SPITM Master / Slave, up to 12 Mbps SMBus TM /12C TM Master / Slave, up to 400 kbps 12C High-Speed Slave, up to 3.4 Mbps 16-bit CRC unit, supporting automatic CRC of flash at 256- byte boundaries 4 Configurable Logic Units Timers/Counters and PWM: - 6-channel programmable counter array (PCA) supporting PWM, capture/compare, and frequency output modes 6 X 16-bit general-purpose timers . Independent watchdog timer, clocked from the low frequen- cy oscillator On-Chip, Non-Intrusive Debugging Full memory and register inspection Four hardware oreakpoints, single-stepping Pre-programmed UART pootloader Temperature range -40 to 85 °C - or -40 to 125 C Available in commercial, industrial, and automotive variants communications and Digital Peripherals:

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

What is the **minimum Voltage Reference Range for DACs**?

The minimum Voltage Reference Range for DACs is 1.15 V.

Source chunks

Chunk 1 (score ≈ 0.02)

Page 31

Parameter	Parameter Symbol Test Condition Min Typ Max Unit !	Symbol	Test Condition	Min	Typ	Max	Unit
Output Voltage	Vout	eel	0	~~	Vop	pov	Output Current

Ask a question …

Deploy

Reset index

Browse

Chat

Chat over indexed PDFs

When you ask a question, the app first **retrieves** the most chunks from the indexed PDFs sections using semantic search, then it **generates** an answer from those top-ranked chunks.

- The assistant's answer appears above.
- The exact chunks that were used are listed underneath.

What are the different **power modes** for EFM8BB3?

The different power modes for the EFM8BB3 are:

- Normal
- Idle
- Suspend
- Snooze
- Shutdown

Source chunks

Chunk 1 (score ≈ 0.02)

Page 3 Part 1

Ask a question …