Automated Extraction and C Code Generation of Event Chart

Mentored By
K Sravani
Scientist 'D'
RCI , DRDO

Mentored By
P Sridhar
Scientist 'E'
ASL, DRDO

Mentored By
A Maruti Sairam
Scientist 'F'
RCI, DRDO

Team Members:

1. A Poornima
2. Sharmila Padamsri
3. Polishetty Ramya Sri
4. Madhumitha K
5. Krishnam Abhina

1. Project Overview

The primary objective of this project is to automate the extraction and translation of event charts into C code.

An **event chart** lists a sequence of events with conditions for execution. Each event is converted into a `case` block within a `switch` statement in C, where the event's condition determines when it runs.

The system parses DOCX event charts, processes logical conditions and control points (DOPs/DIPs), and generates structured, readable C code

2. Source codes

1. `mission_parser.py` - Document parsing and data extraction
2. `c_code_generation.py` - C code generation from parsed data

3. System Requirements and Environment Setup

***Python Version**: Python 3.9 or newer is recommended.*

Required Python Packages:

Core Libraries:

- pip (for package installation)
- pandas
- openpyxl
- python-docx
- PyMuPDF (imported as fitz)
- pdf2docx

UI Libraries:

- tkinter (for file selection interface)
- IPython.display (optional, for inline display in environments like Jupyter)

Standard Libraries:

- os, re, json, tempfile, and sys are used extensively and are bundled with Python

Offline SetUp Instructions:

Copy the following **from the Github to your offline machine**:

- The `wheels/` folder (contains all required `.whl` files)
- The Python scripts:
  - mission_parser.py
  - c_code_generation.py
- The `requirements.txt` file (used to install packages)

Open the terminal or command prompt in the directory where `.whl` files are located, and run:

- pip install --no-index --find-links=wheels -r requirements.txt

4. Prerequisites for Input Documents

- **Uniform Subscript Format**: Maintain consistency in subscript notation. Avoid redundant or varied subscripts (e.g., use a single format like $T_{L1\ SS\ ON\ N}$ and not variations such as $T_{L1\ SS\ N\ ON}$ for the same variable).
- **Parentheses Consistency**: Ensure that logical expressions have properly balanced and clearly defined parentheses for accurate parsing.
- **Subsystem Mapping Flexibility**: The system designer may modify RT-to-subsystem and alias mappings according to specific system requirements.
- **Variable Naming Convention**: Variables prefixed with `T` or `T_` will be automatically converted to `time` in C code during variable declaration.
- **Table Structure**: Must contain columns for Event, Logic Expression, and Event Description, and Remarks
- **Flag Assignment Positioning**: If there is a flag assignment placed above the logical condition, it is treated as a flag set before evaluating the condition for that specific event.
- **Expression Format Guidelines**: Expressions are expected to follow a consistent structure using logical connectors like `AND`, `OR`, and comparison symbols such as `>=`, `<=`, and `==`. Ambiguities or informal syntax should be avoided for successful parsing. Examples provided below illustrate recommended formatting:

- Valid: [T >= (T0 + 0.600 s)] AND (DIP 30 == 1 AND DIP 31 == 0)
- Valid: [(h - h_0) ≥ 8 m] OR (MODECODE == 1 AND T >= (T0 + 1.000 s))
- Invalid: DIP 30 = 1, 31 = 0 → Missing logical operator between clauses.
- Invalid: h - h0 ≥ 8m OR → Trailing operator or incomplete expression.

- **DOP/DIP Format Expectations**: DOP and DIP entries in the remarks column should follow identifiable formats to enable correct parsing. Examples based on parsing logic:
  - Valid: `DOP # 12,13 (N) (ON)` — Basic format with group and status.
  - Valid: `DOP # 12,13 (ON) (N)` — Status and group reversed (also supported).
  - Valid: `DOP # 12,13(N), 87,88(R) (ON)` — Mixed groupings in a single entry.
  - Valid: `DOP # A6,A7(N), B6,B7(R) (ON)` — With alpha-labeled DOPs.
  - Valid: `MIU-A of CCSC DOP # A1,A2 (N) (ON)` — With subsystem alias for RT mapping.
  - Invalid: `12,13 DOP ON R` — Missing `#` symbol and unclear structure.
  - Invalid: `DOP 12, 13 of MIU-A ON` — Lacks required group (N/R) and parentheses.

Consistent formatting ensures that DOPs are correctly assigned to Normal (N) and Redundant (R) groups and mapped to proper subsystems.

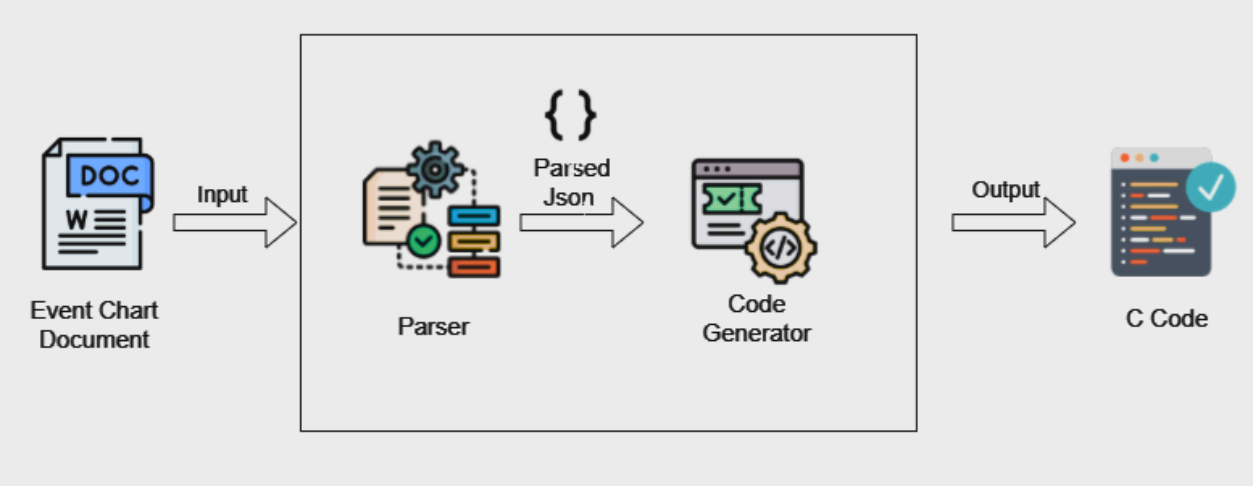
Bus no	Assigned RT nos
Bus_0	RT#1,RT#2,RT#3
Bus_1	RT#4,RT#5,RT#6

RT_no	Subsystem no
RT#1	ss1
RT#2	ss2
RT#3	ss3
RT#4	ss4
RT#5	ss5
RT6	Ss6

5. Structure of the Input Document

Column Name	Description
Event #	A unique identifier for each event (E1 MS PHASE = 1).
Timing Logic	Logical or timing condition to be evaluated, typically involving T variables or expressions. (in secs)
Events	Description of the event.
Remarks	Additional details such as DOP/DIP information, flag assignments, null commands.

6. Architecture



7. Component 1: Event Chart Parser (`mission_parser.py`)

Purpose

Extracts and parses mission logic expressions from technical documents, handling complex nested logical conditions and DOP/DIP control data.

Key Features:

Character Normalization

- **Unicode Handling**: Converts superscript/subscript characters to ASCII equivalents for uniform representation of variables like T_{OOC}→ T_OCC.
- **Greek Letter Support**: Normalizes Greek letters to readable text (α → alpha, β → beta, etc.)
- **Special Character Processing**: Handles mathematical symbols, comparison operators, and units.
- **Unit Stripping:** Units (e.g., `s`, `ms`, `V`, `kg`) are stripped from mathematical expressions to make them C-code compliant. However, the document must be consulted to interpret the physical meaning of such values accurately.

Expression Parsing

- **Logical Operators**: Processes AND/OR operations with proper precedence
- **Nested Expressions**: Handles parentheses, brackets, and complex nesting
- **Bracket Validation**: Comprehensive validation with detailed error reporting
- **Mathematical Expressions**: Distinguishes between descriptive text and mathematical conditions

DOP/DIP Data Handling

- **Extraction**: Parses DOP and DIP control points from the remarks column in documents.
- **Classification**: Separates DOP points into Normal (N) and Redundant (R) groups, associating their respective statuses (ON/OFF).
- **Subsystem and Bus Mapping**: Maps extracted RT numbers or their aliases to specific subsystems and bus numbers.

NOTE: DOP remarks such as `MIU-A of CCSC DOP # A6,A7 (N) (ON)` or `IAU: DOP# 4,5 (N) (ON)` are used to infer subsystem and RT# relationships. Aliases like `MIU-A`, `MIU-B`, `IAU`, and `CPIF` are internally mapped to their corresponding RT numbers. Designers can customize these mappings in the code according to their system's architecture. Incorrect or ambiguous naming may result in `unknown` subsystem or bus number during parsing.

Flag and Null Command Detection

- **Flag Extraction**: Identifies and extracts flag assignments from event descriptions and logic statements.
- **Null Command Detection**: Recognizes explicit instructions to issue null commands in system control sequences.

Output Formats

- JSON (structured, clean, placeholder-based)
    - **parsed_mission_logic_clean.json**: Structured JSON with fully parsed mission logic expressions.
    - **parsed_mission_logic_with_placeholders.json**: Structured JSON with descriptive logic replaced by function-call placeholders.
    - **parsed_dop_dip_by_event.json**: Event-wise structured JSON mapping DOP/DIP data and subsystems.
    - **final_mission_parsing_output.json**: Combined structured JSON file merging mission logic and DOP/DIP data.
- CSV (for quick inspection)
    - **parsed_mission_logic.csv**: Quick-inspection CSV file for parsed mission logic.
- Excel (structured spreadsheet format)
    - **parsed_mission_logic.xlsx**: Excel spreadsheet containing parsed mission logic in structured form

Error Handling

Bracket Validation Errors

- Custom `BracketValidationError` exception class
- Detailed error messages with position information
- Event-specific error tracking
- Automatic error log generation (`bracket_validation_errors.json`)

8. Component 2: C Code Generator (`c_code_generation.py`)

Purpose

Generates executable C code from the structured JSON data (`final_mission_parsing_output.json`) obtained from the mission parser component, enabling automated deployment in embedded and mission-critical systems.

Key Features:

C Code Structuring

- **Macro Definitions `generate_macro_definitions`): Produces macro constants for bus numbers and subsystems to improve readability and maintainability.**.
- **Variable Declaration (`extract_functions_from_conditions`)**: Dynamically declares and initializes float and integer variables based on parsed logic such as timers and status flags that need declaration in the generated code.
- **Logical Condition Conversion**: Converts logical expressions from JSON format into valid C logical conditions using proper operator precedence ensuring correct bracket placement.

Function Generation

- **Placeholder Functions**: Creates placeholder functions for DIP-related conditions and descriptive logic blocks, indicating where user implementation is required.
- **Null Command Functions**: Auto-generates functions for issuing null commands when explicitly stated in the input document.

Automated Commenting and Documentation

- **Multiline Comment Wrapping**: Generates clear, multiline comments describing each event and logical condition for readability and maintainability.

Interface and Initialization Functions

- **Interface Function**: Provides an interface to assign external module values to internal logic variables.
- **Initialization Function**: Auto-generates code to initialize timing and event management variables.

Output

Generated C File: Produces a ready-to-compile `.c` file containing structured logic, variable definitions, and placeholder function implementations tailored for embedded systems and mission control software

9. Limitations

- Parsing PDF documents directly is complex due to their non-uniform internal structures. To overcome this issue, PDF files are first converted to DOCX format internally, ensuring consistent table structure and reliable data extraction. PDF conversion to DOCX can result in potential data loss; developers must manually verify converted documents against original PDFs to ensure accuracy.
- **Initial Script Execution Delay**: When running the scripts for the first time, it's recommended to allow extra time as the tkinter file-selection interface may take a moment to appear. To ensure proper functionality, run and stop the script at least once initially.
- Requires strict adherence to predefined document formats (tabular structure).
- Subsystem mapping depends on explicit RT numbers or predefined alias names.
- Placeholder functions require manual implementation of the actual logic.
- **Scanned PDF documents are not supported**: Documents must be machine-readable. Image-based scanned PDFs (e.g., from scanned printouts) cannot be parsed and will lead to missing or incorrect data extraction.

10. Correctness

The code successfully handles the complex task of parsing mission-critical documents and generating C code, with robust error handling and comprehensive edge case management. The primary concern is the potential for data loss during PDF conversion, which could lead to incomplete or incorrect mission logic. For production use, native DOCX format is strongly recommended, or enhanced PDF parsing capabilities should be implemented.

Performance & Scalability:

Memory Usage

- **Mission Parser**: O(n) where n = document size
- **C Generator**: O(m) where m = number of events
- **Peak Usage**: Reasonable for typical mission documents (< 1000 events)

Processing Speed

- **Small Documents** (< 50 events): Sub-second processing
- **Large Documents** (500+ events): 5-15 seconds (acceptable)
- **Bottlenecks**: PDF conversion is the slowest operation

Validation Points

1. **Input Validation**: File type checking and format validation
2. **Data Integrity**: JSON schema consistency between modules
3. **Output Quality**: Generated C code syntax correctness
4. **Error Propagation**: Meaningful error messages throughout pipeline
5. **PDF Conversion**: Potential data loss not always detected

11. Edge Case Handling

Scenario	Mission Parser	C Generator	Status
Empty expressions	Handled	Handled	Pass
Malformed brackets	Detailed errors	Safe defaults	Pass
Unicode edge cases	Comprehensive mapping	Clean conversion	Pass

Missing data fields	Graceful defaults	NIL handling	Pass
Large documents	Memory usage	Streaming	Acceptable
PDF data loss	**Silent failures possible**	Handles corrupted input	**Critical**

12. Correctness Verification

Mission Parser

- **Unicode Handling**: Tested with complex subscript/superscript combinations
- **Bracket Validation**: Comprehensive test cases for nested structures
- **Operator Precedence**: Verified AND/OR parsing order
- **File Format Support**: Both DOCX and PDF processing validated
- **PDF Conversion**: Limited validation of data preservation during conversion

C Code Generator

- **Variable Extraction**: Correctly identifies float vs int variables
- **Template Generation**: Produces compilable C code structure
- **Function Placeholders**: Generates appropriate function signatures
- **Macro Handling**: Proper bus/subsystem mapping

Critical Limitation: PDF Conversion Data Loss

- **Table Structure Loss**: Complex table layouts may be corrupted or simplified
- **Text Formatting Loss**: Superscripts, subscripts, and special characters may be altered
- **Spatial Relationships**: Precise positioning of elements is not preserved
- **Mathematical Expressions**: Complex formulas may be broken or misinterpreted
- **Image/Diagram Loss**: Embedded graphics and diagrams may be lost or distorted

Edge Cases Handled During Development

- **Inline DOP groupings** like `A6,A7(N), B6,B7(R) (ON):` Logic added to extract multiple DOP groups and apply the common status.
- **Unicode decode issues during file reading:** Handled by explicitly reading `.json` files with `encoding='utf-8'` to avoid `UnicodeDecodeError`.
- **Variables with Subscripts, Superscripts, or Greek Letters:** Introduced a normalization module to map subscripts/superscripts and Greek symbols to ASCII equivalents (e.g., `T₀ → T0`, `α → alpha`).
- **Chained Flag Assignments Within Logic:** Flags such as `flagY = flagA && flagB` embedded within logic were overlooked. Added logic to detect and separate flag assignments, ensuring they're handled properly in C output and initialized independently.

13. Sample Test Case

Test Case 1

Input:

E1 MS PHASE = 1	$T_{BDSep\ N}$ => [T ≥ (T_{OOC} + 0.500 s)]	Pyro normal command is issued for BD separation and diversion through DSM. 500msec delay is given to get sufficient clearance from canister top.	MIU-A of CCSC DOP # A6,A7 (N) (ON) MIU-B of CCSC DOP # B6,B7 (R) (ON)

Parsed Json:


```
[
   {
       "Event Number": "E1",
       "event_description": "Pyro normal command is issued for BD separation
and diversion through DSM. 500msec delay is given to get sufficient clearance
from canister top.",
       "time_assignment": "TBDSep_N",
       "logic": {
           "condition1": "T>=(TOOC + 0.500)"
       },
       "dop_dip": {
```

```
            "busno": "bus_0",
            "subsys": "ss3",
            "dop_n": "A6, A7",
            "dop_r": "B6, B7",
            "status_n": "ON",
            "status_r": "ON",
            "issueNullCommand": "no",
            "flags": "NIL"
        }
    }
]
```

C Code Output:

```c
#define ON 1
#define OFF 0
#define unsigned short int WORD
#define EXTERNVAR
#define LOCALVAR

#define BUS_0 0
#define SS3 3

/* Declare all required variables */
int event;
float timeBDSep_N;

float timeFlight;
float timeOOC;

/* Function prototypes */


/* Placeholder functions for condition checks */


void interface() {
    // timeFlight = external_timeFlight;
    // timeOOC = external_timeOOC;
}

void init_timevar() {
    event = 1;
    timeFlight = 0.0;
    timeBDSep_N = 0.0;
}

#ifdef EXTERNVAR
extern float timeFlight;
extern float timeOOC;
#endif
```

```
void misSeq() {
   WORD dopNos[10];
   WORD dopValues[10];
   interface();

   switch(event) {

       /*
* Event Description: Pyro normal command is issued for BD separation and
diversion
* through DSM. 500msec delay is given to get sufficient clearance from
canister
* top.
*/
       case 1:

           if (timeFlight>=(timeOOC + 0.500)) {

               dopValues[0] = A6;
               dopValues[1] = A7;
               dopValues[2] = B6;
               dopValues[3] = B7;
               misSeqDop(BUS_0, SS3, 4, dopNos, ON);
               timeBDSep_N = timeFlight;
               event = 2;
           }
           break;
       default:
           break;
   }
}
```

Test Case 2

Input:

E2 MS PHASE = 2	Ctrl=1 T_{ooc} => [(T ≥ (T_0 + 0.600 s)) AND (3 consecutive samples @ 10ms of any 2 of the following 4 conditions are true:	Out-Off-Canister is sensed by 4 OOC sensors and declared using 2/4 logic by OBC. For normal pyro channel arming ½ logic is used by taking the status OOC1,OOC2(Diagonally opposite)	Even numbered DIPs are for NO status of respective OOC Snsor, similarly Odd numberered DIPs are NC status of respective OOC Sensor. #Prior to this event OBC should verify the following conditions: DIP 30=0 & DIP 31=1

	DIP 30 = 1 & DIP 31 = 0 (OOC1) DIP 32 = 1 & DIP 33 = 0 (OOC2) DIP 34 = 1 & DIP 35 = 0 (OOC3) DIP 36 = 1 & DIP 37 = 0 (OOC4) Of CCSC using following logic) AND ((h - h_0) ≥ 8 m)] OR [MODECODE = 1 AND T ≥ (T_0+1.000 s)]	For Redundant pyro channel arming ½ logic is used by taking the status of OOC3,OOC4 H0 is the Lift-off height.	(OOC 1) DIP 32 = 1 & DIP 33 = 0 (OOC2) DIP 34 = 1 & DIP 35 = 0 (OOC3) DIP 36 = 1 & DIP 37 = 0 (OOC4) Of CCSC. Otherwise, the OOC Sensor not meeting above condition should be declared invalid, and not to be considered for OOC sensing. Low to high transition occurs for NO status and High to low transition occurs for NC status at the time of Out of Canister sensing.

Parsed Json:


```
[
   {
       "Event Number": "E2",
       "event_description": "Out-Off-Canister is sensed by 4 OOC sensors and
declared using 2/4 logic by OBC. For normal pyro channel arming ½ logic is
used by taking the status OOC1,OOC2(Diagonally opposite) For Redundant pyro
channel arming ½ logic is used by taking the status of OOC3,OOC4 H0 is the
Lift-off height.",
       "time_assignment": "TOOC",
       "logic": {
           "flags_inside": [
               "Ctrl=1"
           ],
           "condition1": "T>=(T0 + 0.600)",
           "operator1": "**and",
           "condition2": "processDips1Event2()  /* 3 consecutive samples @ 10
of DIP 30==1 & DIP 31==0 (OOC1) DIP 32==1 & DIP 33==0 (OOC2) DIP 34==1 & DIP
35==0 (OOC3) DIP 36==1 & DIP 37==0 (OOC4) Of CCSC using following logic */",
           "operator2": "**and",
           "condition3": "(h - h0)>=8",
           "operator3": "*or",
           "condition4": "MODECODE==1",
           "operator4": "**and",
           "condition5": "T>=(T0+1.000)"
```

```
        },
        "dop_dip": {
            "busno": "bus_0",
            "subsys": "ss3",
            "dop_n": "NIL",
            "dop_r": "NIL",
            "status_n": "NIL",
            "status_r": "NIL",
            "issueNullCommand": "no",
            "flags": "NIL"
        }
    }
]
```


C Code Output:

```c
#define ON 1
#define OFF 0
#define unsigned short int WORD
#define EXTERNVAR
#define LOCALVAR

#define BUS_0 0
#define SS3 3

/* Declare all required variables */
int event;
float timeOOC;

float MODECODE;
float h;
float h0;
float time0;
float timeFlight;
int Ctrl;

/* Function prototypes */
int processDips1Event2(); // TODO: Define this function - currently only
declared

/* Placeholder functions for condition checks */

/* processDips1Event2
* Purpose: 3 consecutive samples @ 10 of DIP 30==1 & DIP 31==0 (OOC1) DIP 32==1
& DIP 33==0 (OOC2) DIP 34==1 & DIP 35==0 (OOC3) DIP 36==1 & DIP 37==0 (OOC4) Of
CCSC using following logic
*/
bool processDips1Event2() {
```

```
   // 3 consecutive samples @ 10 of DIP 30==1 & DIP 31==0 (OOC1) DIP 32==1 & DIP
33==0 (OOC2) DIP 34==1 & DIP 35==0 (OOC3) DIP 36==1 & DIP 37==0 (OOC4) Of CCSC
using following logic
   // return 1;
}

void interface() {
   // Ctrl = external_Ctrl;
   // MODECODE = external_MODECODE;
   // h = external_h;
   // h0 = external_h0;
   // time0 = external_time0;
   // timeFlight = external_timeFlight;
}

void init_timevar() {
   event = 1;
   timeFlight = 0.0;
   timeOOC = 0.0;
}

#ifdef EXTERNVAR
extern float time0;
extern float MODECODE;
extern float h;
extern float timeFlight;
extern float h0;
extern int Ctrl;
#endif

void misSeq() {
   WORD dopNos[10];
   WORD dopValues[10];
   interface();

   switch(event) {

      /*
* Event Description: Out-Off-Canister is sensed by 4 OOC sensors and declared
* using 2/4 logic by OBC. For normal pyro channel arming • logic is used by
taking
* the status OOC1,OOC2(Diagonally opposite) For Redundant pyro channel arming •
* logic is used by taking the status of OOC3,OOC4 H0 is the Lift-off height.
*/
      case 2:
                Ctrl = 1;
          if ((((timeFlight>=(time0 + 0.600)) && (processDips1Event2())) && ((h
- h0)>=8)) || ((MODECODE==1) && (timeFlight>=(time0+1.000)))) {



              timeOOC = timeFlight;
              event = 3;
```

```
            }
            break;
        default:
            break;
    }
}
```

Test Case 3

Input:

E3 MS PHASE = 3	$T_{GFUF\ ON\ N}$=> [T ≥ (T_{OOC} +0.200s+ $T_{GFUF\ Delay}$)] $T_{GFUF\ Delay}$ calculated by guidance for platform launch(ref. Appendix B) $T_{GFUF\ Delay}$ = 0.000 for Ground launch $T_{GFUF\ Delay}$ = 0.100 for Pontoon Launch	Firing of Frid Fin(GF) unfolding through normal.Adaptive GFUF delay disabled in group/pontoon launches and small fixed delay used to unfold GF early in air/water as there is no significant missile attitude divergence is expected in these missions..	BS1 MIU-1(C2), (RT#03): DOP # 60, 64, 65 (ON) (N)

Parsed Json:


```
[
    {
        "Event Number": "E3",
        "event_description": "Firing of Frid Fin(GF) unfolding through normal.
Adaptive GFUF delay disabled in group/pontoon launches and small fixed delay
used to unfold GF early in air/water as there is no significant missile
attitude divergence is expected in these missions..",
        "time_assignment": "TGFUF_ON_N",
        "logic": {
            "condition1": "condition1forEvent3()  /* (T>=(TOOC +0.200+
TGFUF_Delay )) TGFUF_Delay_calculated_by_guidance_for_platform launch(ref.
Appendix B) TGFUF_Delay==0.000 for Ground launch TGFUF_Delay==0.100 for
Pontoon Launch */"
        },
        "dop_dip": {
            "busno": "bus_0",
            "subsys": "ss3",
```

```
            "dop_n": "60, 64, 65",
            "dop_r": "NIL",
            "status_n": "ON",
            "status_r": "NIL",
            "issueNullCommand": "no",
            "flags": "NIL"
        }
    }
]
```

C Code Output:

```c
#define ON 1
#define OFF 0
#define unsigned short int WORD
#define EXTERNVAR
#define LOCALVAR

#define BUS_0 0
#define SS3 3

/* Declare all required variables */
int event;
float timeGFUF_ON_N;



/* Function prototypes */
int condition1forEvent3(); // TODO: Define this function - currently only
declared

/* Placeholder functions for condition checks */

/* condition1forEvent3
* Purpose: (timeFlight>=(timeOOC +0.200+ timeGFUF_Delay ))
timeGFUF_Delay_calculated_by_guidance_for_platform launch(ref. Appendix B)
timeGFUF_Delay==0.000 for Ground launch timeGFUF_Delay==0.100 for Pontoon
Launch
*/
int condition1forEvent3() {
    // (timeFlight>=(timeOOC +0.200+ timeGFUF_Delay ))
timeGFUF_Delay_calculated_by_guidance_for_platform launch(ref. Appendix B)
timeGFUF_Delay==0.000 for Ground launch timeGFUF_Delay==0.100 for Pontoon
Launch
    // return 1;
}

void interface() {

}

void init_timevar() {
```

```
   event = 1;
   timeFlight = 0.0;
   timeGFUF_ON_N = 0.0;
}

#ifdef EXTERNVAR

#endif

void misSeq() {
   WORD dopNos[10];
   WORD dopValues[10];
   interface();

   switch(event) {

       /*
* Event Description: Firing of Frid Fin(GF) unfolding through normal. Adaptive
* GFUF delay disabled in group/pontoon launches and small fixed delay used to
* unfold GF early in air/water as there is no significant missile attitude
* divergence is expected in these missions..
*/
       case 1:

           if (condition1forEvent3()) {

               dopValues[0] = 60;
               dopValues[1] = 64;
               dopValues[2] = 65;
               misSeqDop(BUS_0, SS3, 3, dopNos, ON);
               timeGFUF_ON_N = timeFlight;
               event = 2;
           }
           break;
       default:
           break;
   }
}
```

14. Summary

This project presents a robust pipeline for extracting mission logic from structured DOCX documents and generating corresponding C code tailored for embedded mission control systems. It encompasses two main components—**mission_parser.py** and **c_code_generation.py**—that together enable end-to-end transformation from natural-language logic tables to C code. The parser handles normalization, nested logic parsing, flag recognition, and DOP/DIP interpretation, while the code generator builds structured, maintainable C code with placeholder functions for further customization.

15. Conclusion

The system effectively bridges the gap between high-level mission documentation and low-level embedded code. It handles most edge cases in logic syntax, document structure, and variable mapping. The output is well-structured, with autogenerated macros, function placeholders, and initialization routines. While the current setup supports DOCX files, PDF conversion introduces critical limitations, especially with complex formatting or scanned content. This tool significantly reduces manual translation effort and improves reliability in the development of mission-critical systems.

16. Future Work

Future development will focus on improving flexibility, robustness, and input support. A key goal is enabling **native PDF parsing** without relying on external file conversions, which often lead to data loss or formatting issues. Libraries like PyMuPDF or pdfplumber will be explored to extract structured content such as tables and logic blocks directly from PDF files.

Additionally, to support scanned mission documents, **OCR integration** is planned using tools like Tesseract. This would allow the parser to handle image-based documents where text is not embedded.