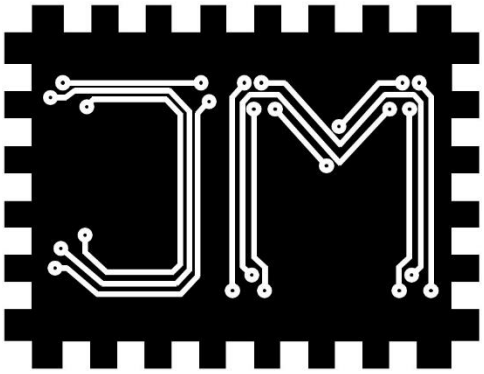

Jeff’s Laboratory

NMSW01 – Flight Manager Sequencer

Comments	Revision	Date	Author
Initial Release	A	December 27, 2024	J. Mays

1. References

1. NESW02 – Flight Manager Events
2. NESW04 - Flight Manager Local Loop
3. NESW03 – Estimator

2. Purpose

This document describes the SEQUENCER software capability (SWC) as illustrated in Matlab/Simulink, and then further discussed in C/C++ software. SEQUENCER uses states from the EVENTS [1], CMD.TVC, CMD.AFTFIN [2], and ESTIMATOR [3] to derive flight sequence changes.

The original implementation was written in Matlab/Simulink and then derived into C/C++ software for compilation onto hardware. Custom Simulink and C/C++ libraries were created to align the two languages.

3. Design Description

The SEQUENCER SWC is the main state machine software for the New Mays model rocket. The state machine initializes in the Idle state and can move to other states given manual commands or other SWC outputs. The Sequencer has a high- and low-level state machine output: Segment and State. The segment is a high-level sequencer that describes the system's current configuration. The state is a subset of the segment. This is mainly applicable in BIT, since there are various tests to choose from while in the BIT segment. All segments and states are described in Table 1 and Table 2.

Table 1: Segments

Segment	Value	Description
Idle	1	Default segment. Everything deactivated.
BIT	2	Built-in-Test segment used for testing and calibration of vehicle components.
Passthrough	4	Segment where all manual commands are allowed to overwrite commands sent internally from software. This allows for testing to be performed on the ground.
Countdown	8	Auto sequence start, where the system is monitoring health and counting down to a known launch time.
Launch	16	Once Countdown goes to 0, we transition to Launch. This segment is where the pyro charges within the solid rocket motor are ignited, and the software is configured to be ready for a liftoff event.
Boost	32	Once liftoff is detected from the EVENTS SWC, we transition to Boost. This segment is used to denote that the rocket motor is on and that we are ascending.
Coast	64	Segment that comes right after Boost via detection of the rocket motor burning out.
Reentry	128	Once we are traveling back towards the ground, we transition into this segment.
Under Chutes	256	Once certain criteria are met, we transition to this segment and deploy the chutes. We are in this segment until we reach the ground.
Touchdown	512	Segment denoting that we are on the ground after being under chutes.
Pad Abort	16384	During Countdown, if certain criteria aren't met, we will pad abort, keeping the launch from occurring.
Flight Abort	32768	During ascent, if the vehicle detects that it is not following the guidance profile, it will abort mid-flight, causing its chutes to deploy. This is a safety measure to protect observers and property.

Table 2: BIT States

State	Value	Description
nichts	0	Nichts means nothing in German.
bit_tvc	1	TVC profile test. This test must be completed before we are allowed to launch.
bit_aftfin	2	Aft Fin profile test. This test must be completed before we are allowed to launch.
bit_sns	3	Sensor test that calibrates the accelerometer and gyroscope. This test must be completed before we are allowed to launch.

4. Interface Control Document

The SEQUENCER SWC input and output buses are shown below. This SWC is called by the FLIGHT_MANAGER, and its elements are populated to the vehicle state vector every cycle count.

Table 3: SEQUENCER input bus

App	Direction	Hierarchy	Element	DataType	Rows	Columns	Comment
fm	in	ops.seq	segment_cmd	uint16_t	1	1	Ops Segment Command
fm	in	ops.seq	state_cmd	uint16_t	1	1	Ops Mode Command (Sequencer state mode)
fm	out	cmd.tvc	bit_complete	bool	1	1	TVC BIT complete flag
fm	out	cmd.aftfin	bit_complete	bool	1	1	Aft Fin BIT complete flag
fm	out	est.nav	bit_complete	bool	1	1	Sensor BIT complete
fm	out	events	liftoff	bool	1	1	Liftoff event indicator
fm	out	events	burnout	bool	1	1	SRM burnout event indicator
fm	out	events	reentry	bool	1	1	Reentry event indicator
fm	out	events	touchdown	bool	1	1	Touchdown event indicator
fm	out	events	pad_abort	bool	1	1	Pad Abort limits have been exceeded
fm	out	events	flt_abort	bool	1	1	Flight Abort limits have been exceeded

Table 4: SEQUENCER output bus

App	Direction	Hierarchy	Element	DataType	Rows	Columns	Comment
fm	out	seq	segment	uint16_t	1	1	FM sequencer state
fm	out	seq	state	uint16_t	1	1	FM state state
fm	out	seq	enable	uint16_t	1	1	Enable flag for downstream SWC (not used)
fm	out	seq	launch_tgo	float	1	1	Time down till launch segment
fm	out	seq	time	float	1	1	Time within the current segment

5. Pre-Configured Gains

Each SWC has an initialization subroutine that initializes parameterized gains. The gains are first defined in Matlab, and then are autocoded into a Cpp file for reference during compilation of the Cpp version of this SWC. The following code snip shows the gain subroutine for the SEQUENCER.

```
function [gains] = init_gains_seq(config)

% Load common gains
common = init_gains_common();

%% Countdown
% Countdown will last 15 seconds. Subtract one tick.
gains.countdown.time = (15 * C_SEC) - common.timing.dt;

%% Launch detection
% Accel threshold that the sequencer uses to detect when we liftoff
```

```
gains.launch.boost_acc_threshold = 15 * C_M/C_SEC/C_SEC;

% Create a timeout such that we can sufficiently test that the engine is
% not going to launch and then abort/safe the vehicle
gains.launch.timeout = (10 * C_SEC) - common.timing.dt; % 10 sec timeout

%% Boost detection
% Burnout detection is now resolved to the EVENTS SWC

%% Flight Abort
% FLT-201: after 2 seconds of being in flight abort, transition back to idle
gains.flt_abort.timeout = (2 * C_SEC) - common.timing.dt; % 2 sec timeout

end
```

6. Software Logic

A high-level capture of the `FLIGHT_MANAGER` software logic, as it is defined in Simulink, is shown in Figure 1. The `SEQUENCER` software capability is highlighted.

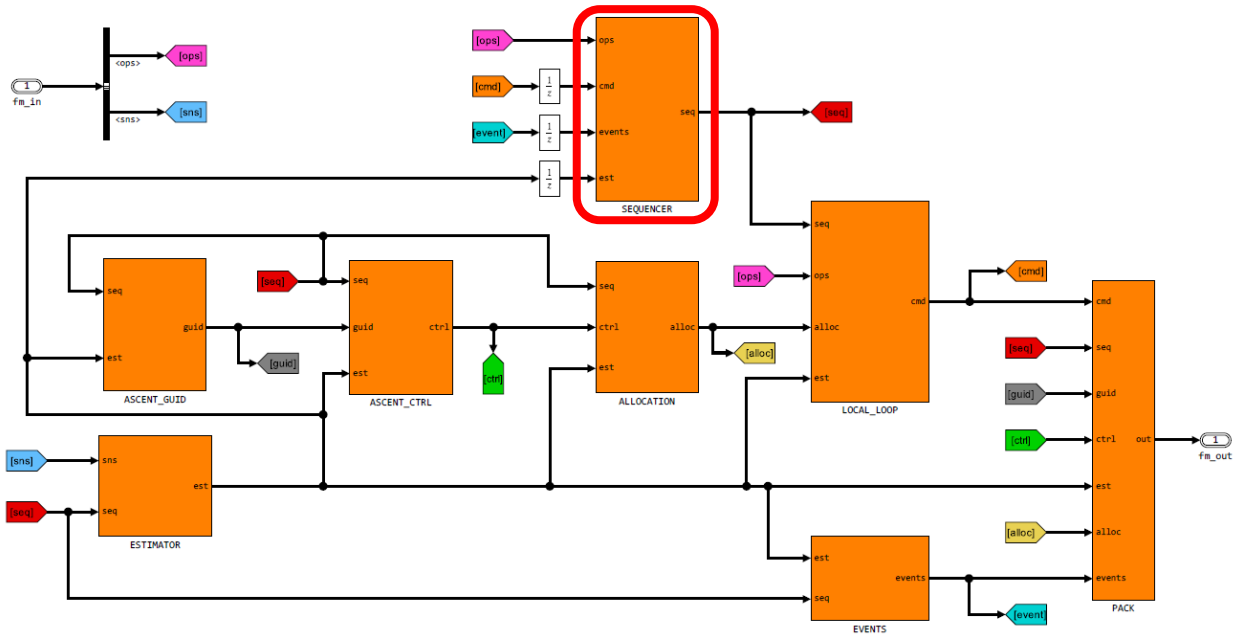

Figure 1: Flight Manager

Under the mask of the `SEQUENCER` block in Figure 1 is the Simulink logic shown in Figure 2. This software mainly consists of a Matlab Function block to perform the sequencing logic.

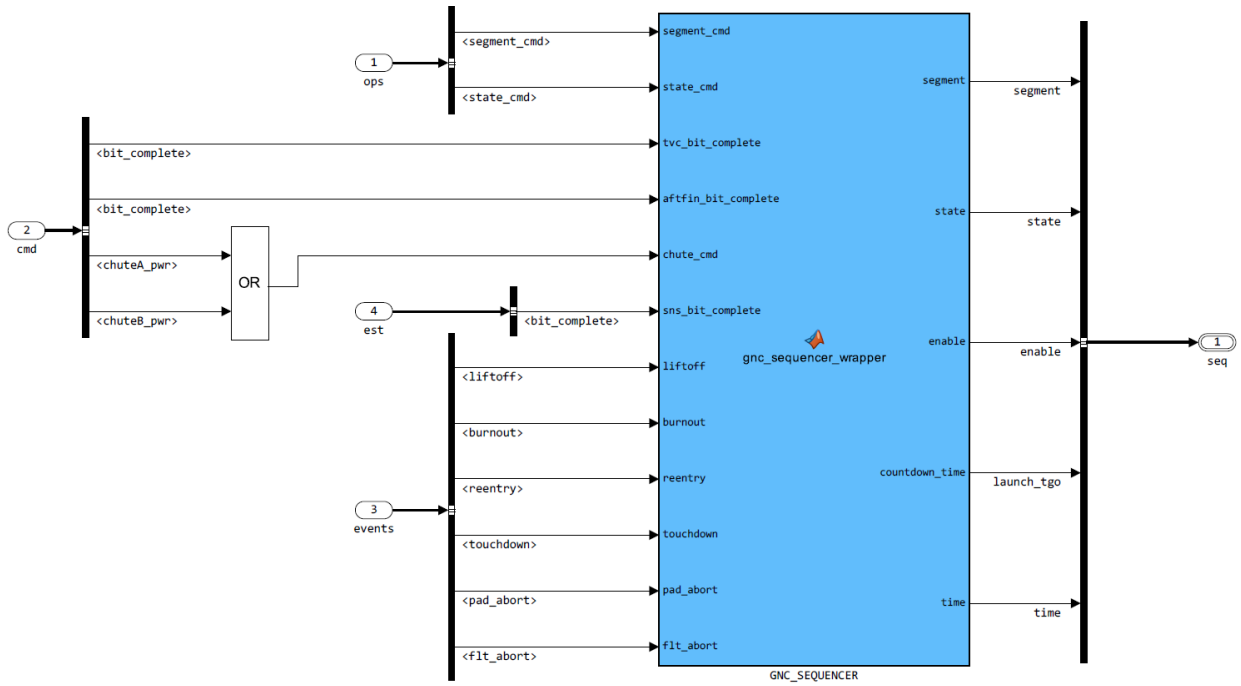

Figure 2: SEQUENCER software capability

The above GNC_SEQUENCER is defined in the Matlab scripting below. The sequencer starts off in the IDLE state and then is allowed to transition to other states as certain criteria is met. The reader should use this set of scripting to infer sequencing behavior.

Note that while the following code is presented in Matlab scripting, the actual code deployed to the vehicle is written in C/C++. The below Matlab code was used as a development tool to get the desired logic working quickly and efficiently. Once working as intended, the C/C++ code was written to follow the same logic using the C/C++ private functions I developed. After converting, a unit test was developed to ensure matching logic. One of the comparisons of this test is shown in Figure 3.

```
function [segment, state, segment_time, countdown_time, time] = gnc_sequencer(segment_cmd, state_cmd,
tvc_bit_complete, aftfin_bit_complete, sns_bit_complete, liftoff, burnout, reentry, ...
                                        chute_cmd, touchdown, pad_abort, flt_abort, COMMON, SEQ, DT)
% GNC Sequencer logic for New Mays
%
% Author: Jeff Mays

% Setup persistent variables
persistent segment_ state_ segment_time_ prev_segment_
persistent prev_segment_cmd_ prev_state_cmd_
persistent time_ countdown_time_
persistent bit_lockout

% Load in segment defines from a common script/location
seq         = fm_sequencer_states();
IDLE        = seq.segment.idle;
BIT         = seq.segment.bit;
PASSTHROUGH = seq.segment.passthrough;
COUNTDOWN   = seq.segment.countdown;
LAUNCH      = seq.segment.launch;
BOOST       = seq.segment.boost;
COAST       = seq.segment.coast;
REENTRY     = seq.segment.reentry;
CHUTES      = seq.segment.under_chutes;
TOUCHDOWN   = seq.segment.touchdown;
PAD_ABORT   = seq.segment.pad_abort;
FLT_ABORT   = seq.segment.flt_abort;

% states
```

```
state_nichts     = seq.state.nichts;
state_bit_tvc    = seq.state.bit_tvc;
state_bit_aftfin = seq.state.bit_aftfin;
state_bit_sns    = seq.state.bit_sns;

% If first run through, init outputs
if (isempty(segment_))
    segment_          = IDLE;
    prev_segment_     = uint16(0);
    state_            = state_nichts;
    segment_time_     = 0.0;
    prev_segment_cmd_ = uint16(0);
    prev_state_cmd_   = uint16(0);
    time_             = 0.0;
    bit_lockout       = false;
    countdown_time_   = 999.0;
end

% If the segment command hasnt changed, null the command
if (segment_cmd == prev_segment_cmd_)
    segment_cmd = uint16(0);
end

% If the state command hasnt changed, cancel the command
if (state_cmd == prev_state_cmd_)
    state_cmd = uint16(0);
end

% Time
time_ = time_ + DT;

%% Segment Changes

switch segment_

    case IDLE
        segment_time_ = segment_time_ + DT;
        if (segment_cmd == BIT)
            segment_ = BIT;
            state_   = state_nichts;
        elseif (segment_cmd == PASSTHROUGH)
            segment_ = PASSTHROUGH;
            state_   = state_nichts;
        elseif (segment_cmd == COUNTDOWN && tvc_bit_complete && aftfin_bit_complete && sns_bit_complete)
            segment_ = COUNTDOWN;
            state_   = state_nichts;
        end

    case BIT
        segment_time_ = segment_time_ + DT;
        if (segment_cmd == IDLE && bit_lockout == false)
            % Only when the BIT completes will the segment be allowed to
            % transition away from BIT
            segment_ = IDLE;
            state_   = state_nichts;

        elseif (tvc_bit_complete == true && state_ == state_bit_tvc) || ...
           (aftfin_bit_complete == true && state_ == state_bit_aftfin) || ...
           (sns_bit_complete == true && state_ == state_bit_sns)
            state_      = state_nichts; % Leave the test bit state
            bit_lockout = false;

        elseif (state_cmd == state_bit_tvc && bit_lockout == false)
            bit_lockout = true;
            state_      = state_bit_tvc;

        elseif (state_cmd == state_bit_aftfin && bit_lockout == false)
            bit_lockout = true;
            state_      = state_bit_aftfin;

        elseif (state_cmd == state_bit_sns && bit_lockout == false)
            bit_lockout = true;
            state_      = state_bit_sns;
        end

    case PASSTHROUGH
        segment_time_ = segment_time_ + DT;
        if (segment_cmd == IDLE)
```

```
            segment_ = IDLE;
            state_   = state_nichts;
        end

    case COUNTDOWN
        segment_time_ = segment_time_ + DT;
        countdown_time_ = SEQ.countdown.time - segment_time_;

        if (segment_cmd == IDLE)
            segment_        = IDLE;
            state_          = state_nichts;
            countdown_time_ = 999.0; % Reset timer

        elseif (segment_cmd == PAD_ABORT || pad_abort)
            segment_        = PAD_ABORT;
            state_          = state_nichts;
            countdown_time_ = 999.0; % Reset timer

        elseif (countdown_time_ < 0)
            segment_ = LAUNCH;
            state_   = state_nichts;
        end

    case LAUNCH
        segment_time_ = segment_time_ + DT;
        if liftoff
            segment_ = BOOST;
            state_   = state_nichts;

        elseif flt_abort
            segment_ = FLT_ABORT;
            state_   = state_nichts;

        elseif (segment_time_ > SEQ.launch.timeout)
            segment_ = PAD_ABORT;
            state_   = state_nichts;
        end

    case BOOST
        segment_time_ = segment_time_ + DT;
        if burnout
            segment_ = COAST;
            state_   = state_nichts;

        elseif flt_abort
            segment_ = FLT_ABORT;
            state_   = state_nichts;
        end

    case COAST
        segment_time_ = segment_time_ + DT;
        if reentry
            segment_ = REENTRY;
            state_   = state_nichts;

        elseif flt_abort
            segment_ = FLT_ABORT;
            state_   = state_nichts;
        end

    case REENTRY
        segment_time_ = segment_time_ + DT;
        if chute_cmd
            segment_ = CHUTES;
            state_   = state_nichts;
        end

    case CHUTES
        segment_time_ = segment_time_ + DT;
        if touchdown
            segment_ = TOUCHDOWN;
            state_   = state_nichts;
        end

    case TOUCHDOWN
        segment_time_ = segment_time_ + DT;
        % Do nothing. We want to have to power cycle the vehicle before we
        % can get back into idle (resets SW states)
```

```
    case PAD_ABORT
        segment_time_ = segment_time_ + DT;
        % Do nothing, make OPS command out of pad abort
        if (segment_cmd == IDLE)
            segment_    = IDLE;
            state_      = state_nichts;
        end

    case FLT_ABORT
        segment_time_ = segment_time_ + DT;
        if (segment_time_ > SEQ.flt_abort.timeout)
            segment_ = IDLE;
            state_   = state_nichts;
        end

end

%% Reset time every time we are in a new segment
if (segment_ ~= prev_segment_)
    segment_time_ = 0.0;
    prev_segment_ = segment_;
end

%% Set old
if (segment_cmd > 0)
    prev_segment_cmd_ = segment_cmd;
end
if (state_cmd > 0)
    prev_state_cmd_ = state_cmd;
end

%% Set outputs
segment        = segment_;
state          = state_;
segment_time   = single(segment_time_);
countdown_time = single(countdown_time_);
time           = single(time_);

end
```

7. Software Validation & Unit Tests

> ***Justification for Lack of Unit Test Development:*** *This project is* ***for-fun****. It is not meant to represent the actions that would be taken in a fully staffed and/or a human safety critical aerospace project/program. Rather than a long and cumbersome unit test development and documentation process of each SWC, desktop simulations were used to provide evidence of sufficient behavior out of this particular SWC. While not all possible code coverage avenues are explicitly checked in the nominal monte-carlo simulations, all requirements are enforced in post processing. This is acceptable for the level of rigor required for this project.*

The following plots are shown from a nominal simulation where the Simulink version is used. The C/C++ software was then compiled, and the same input buses used were passed through the shared object for comparison in the Matlab environment. The following plot shows the relationship between the two versions.

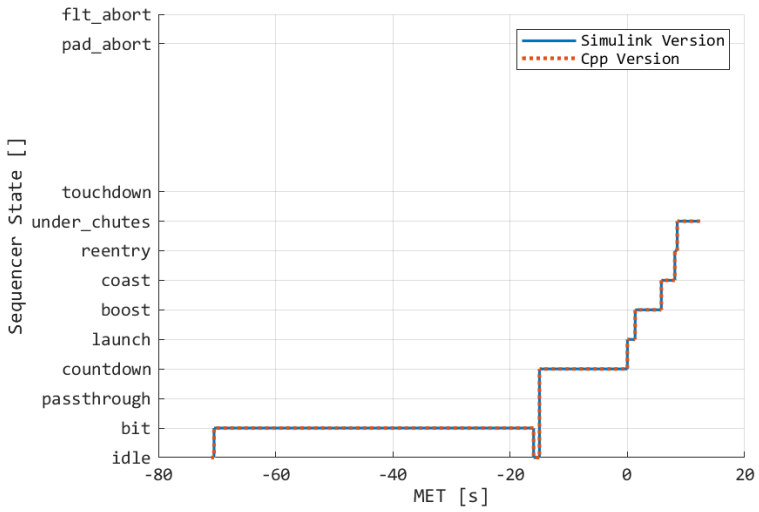

Figure 3: Segment Comparison (note that there is a simulation issue here where the touchdown segment is not reached, this is known as was fixed)