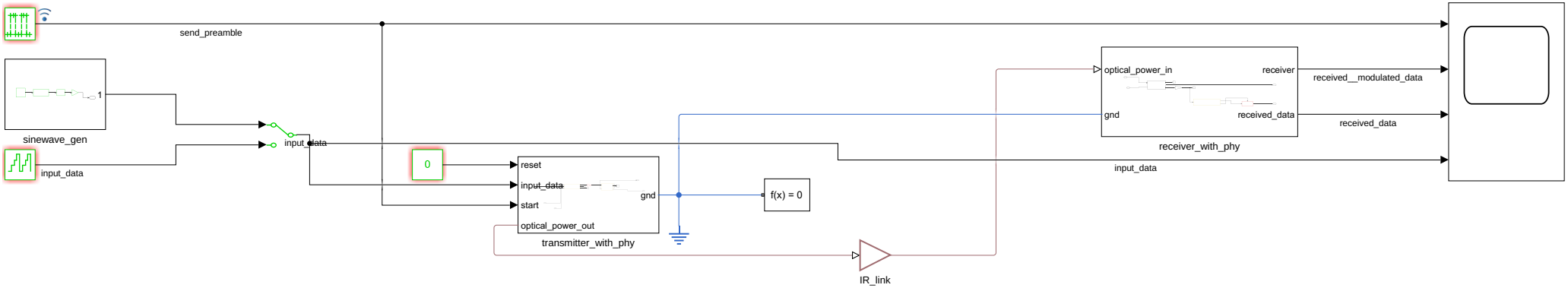
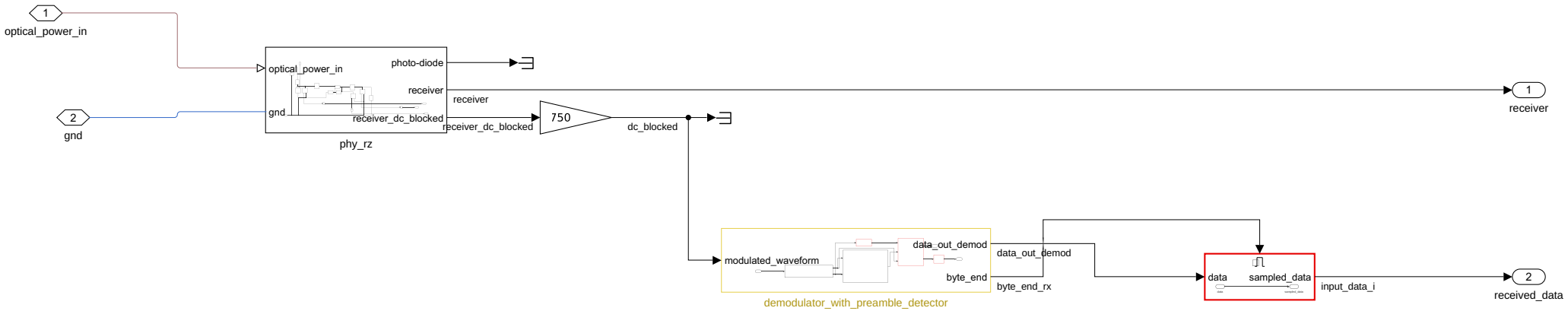
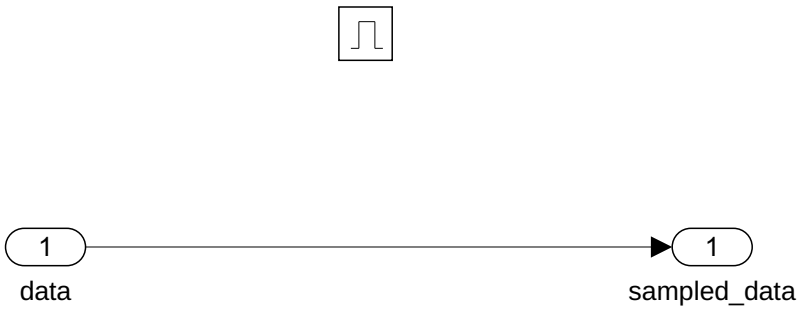
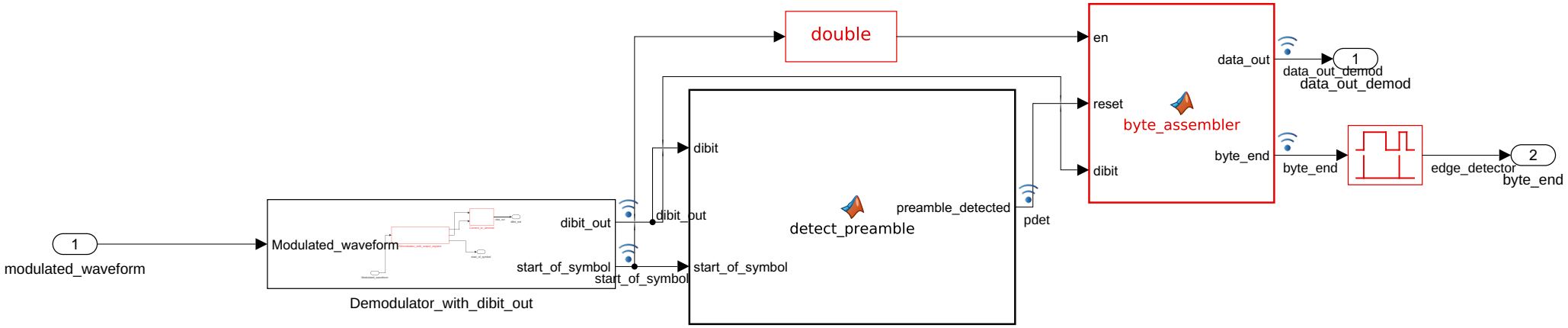
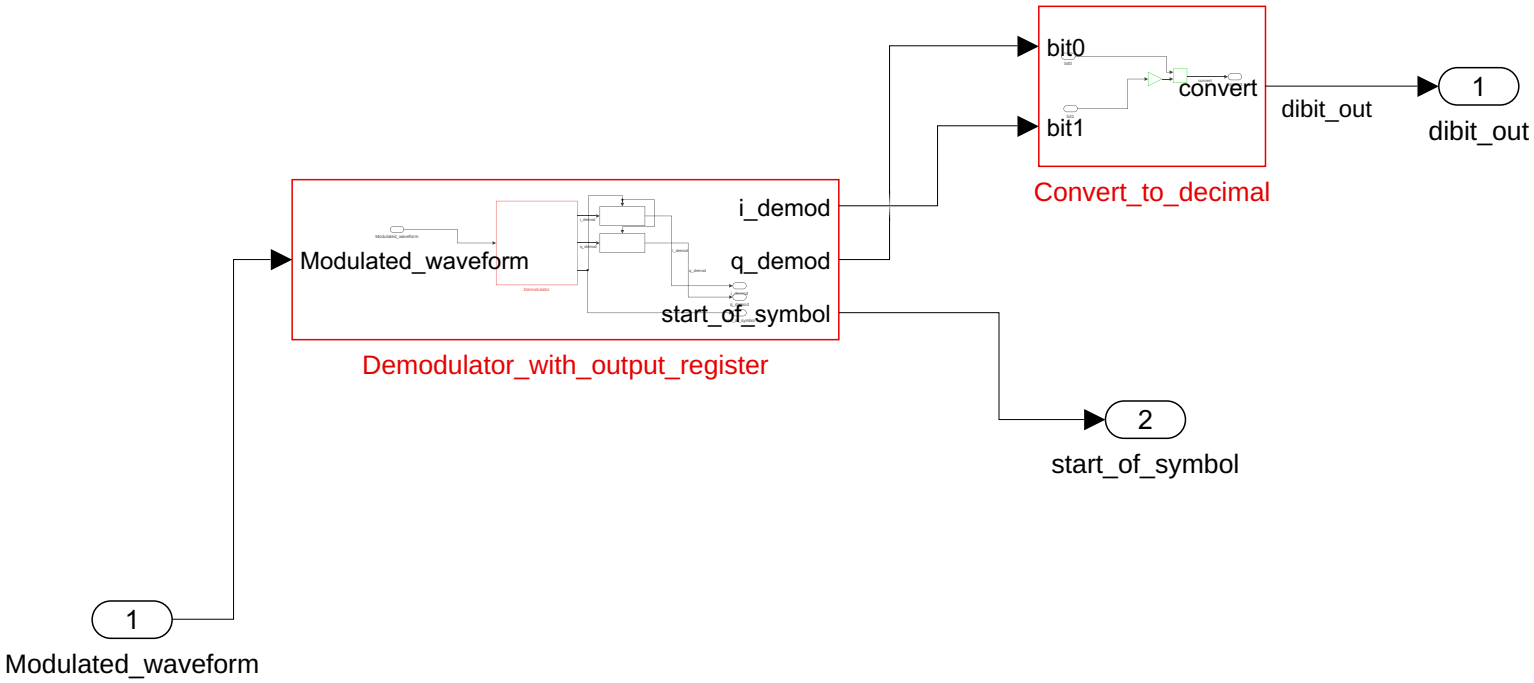
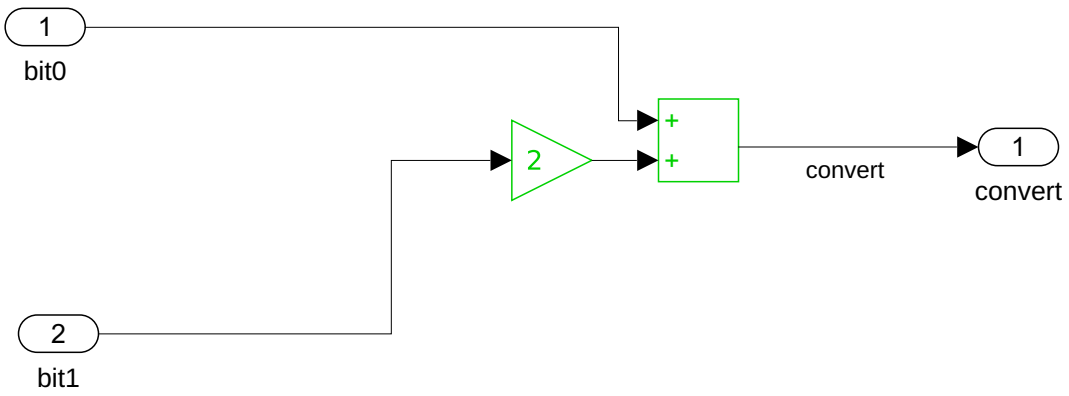

send_preamble
1
sinewave_gen
input_data
input_data
0
reset
input_data
start
optical_power_out
gnd
transmitter_with_phy
f(x) = 0
IR_link
optical_power_in
gnd
receiver
received_data
receiver_with_phy
received__modulated_data
received_data
input_data

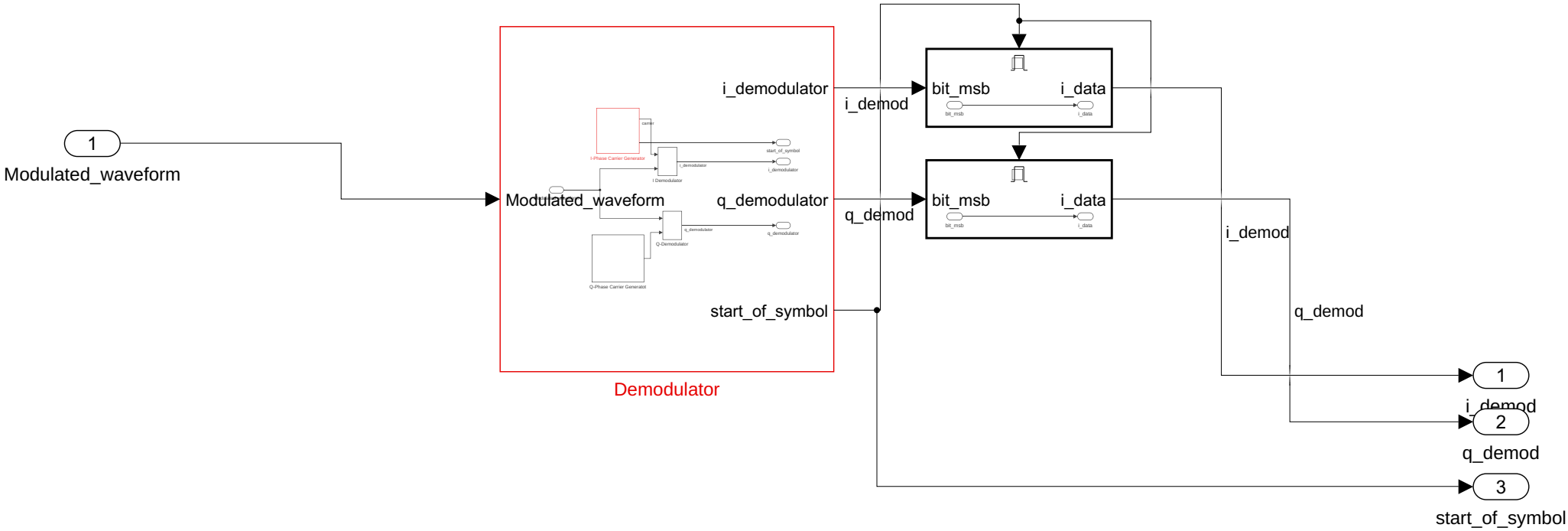

1
optical_power_in
2
gnd
optical_power_in
gnd
photo-diode
receiver
receiver_dc_blocked
phy_rz
receiver
receiver_dc_blocked
750
dc_blocked
1
receiver
modulated_waveform
data_out_demod
byte_end
demodulator_with_preamble_detector
data_out_demod
byte_end_rx
data
sampled_data
input_data_i
2
received_data

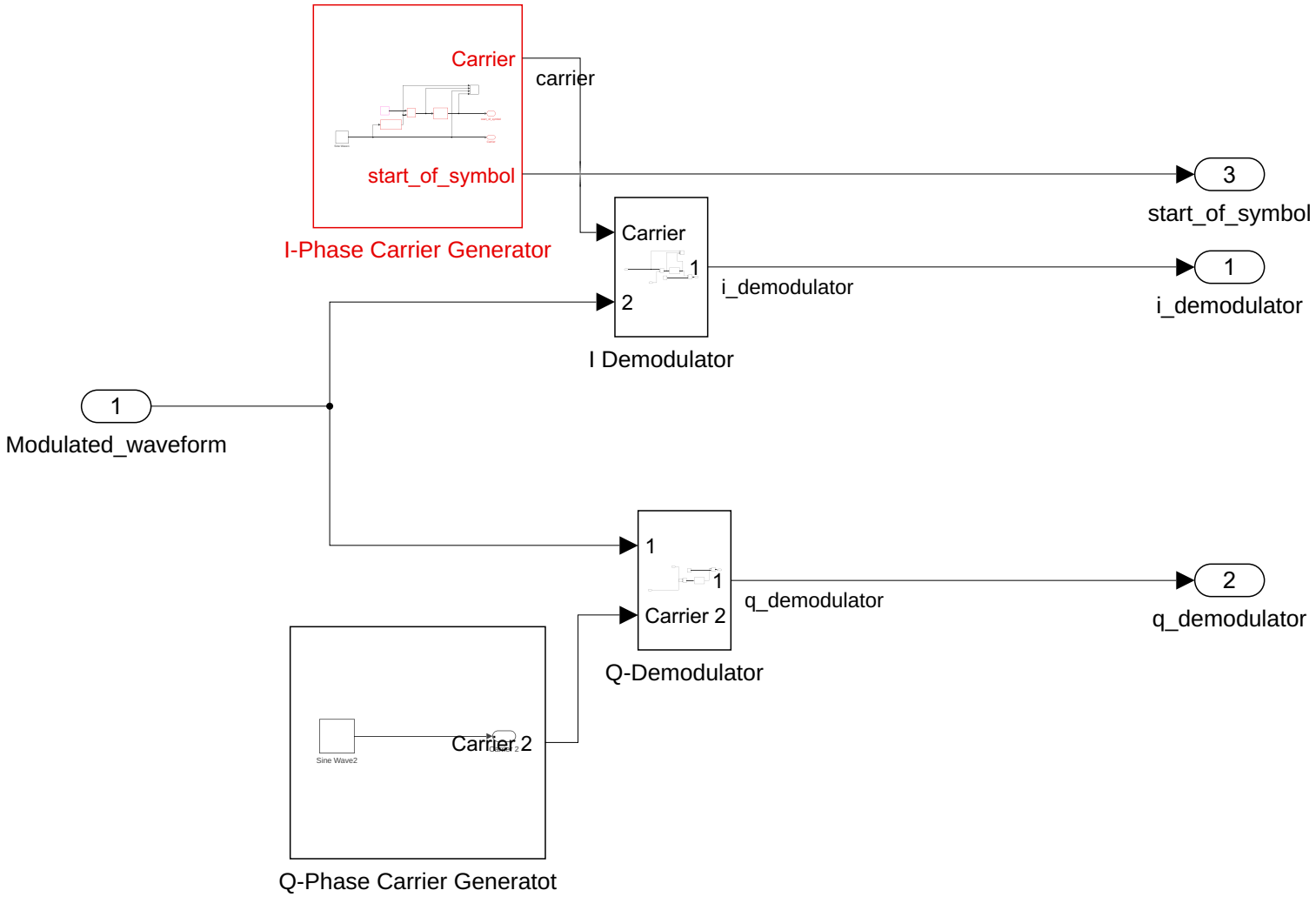

1
data
1
sampled_data

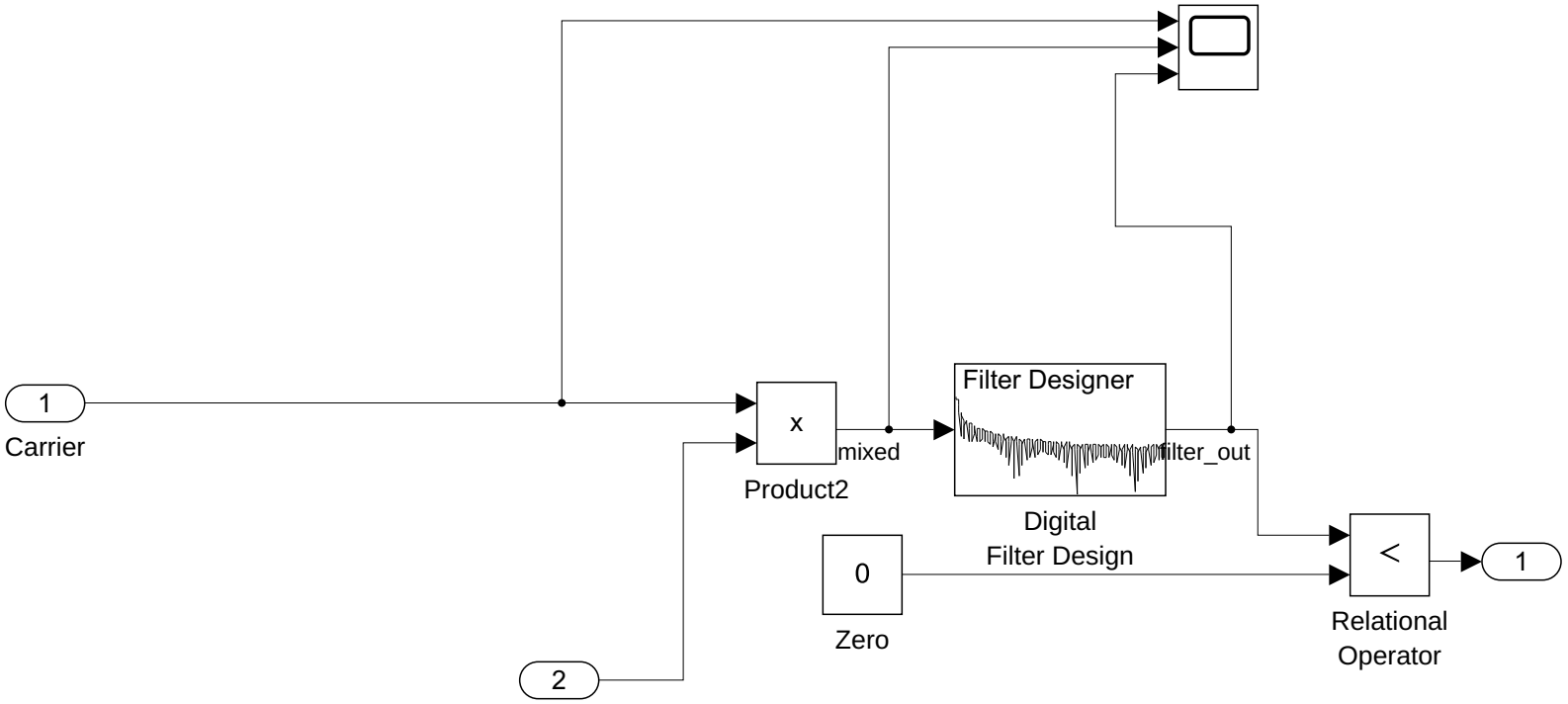

1
modulated_waveform
Modulated_waveform
dibit_out
start_of_symbol
Demodulator_with_dibit_out
dibit_out
start_of_symbol
double
dibit
start_of_symbol
detect_preamble
preamble_detected
pdet
en
reset
dibit
byte_assembler
data_out
byte_end
1
data_out_demod
data_out_demod
byte_end
edge_detector
2
byte_end

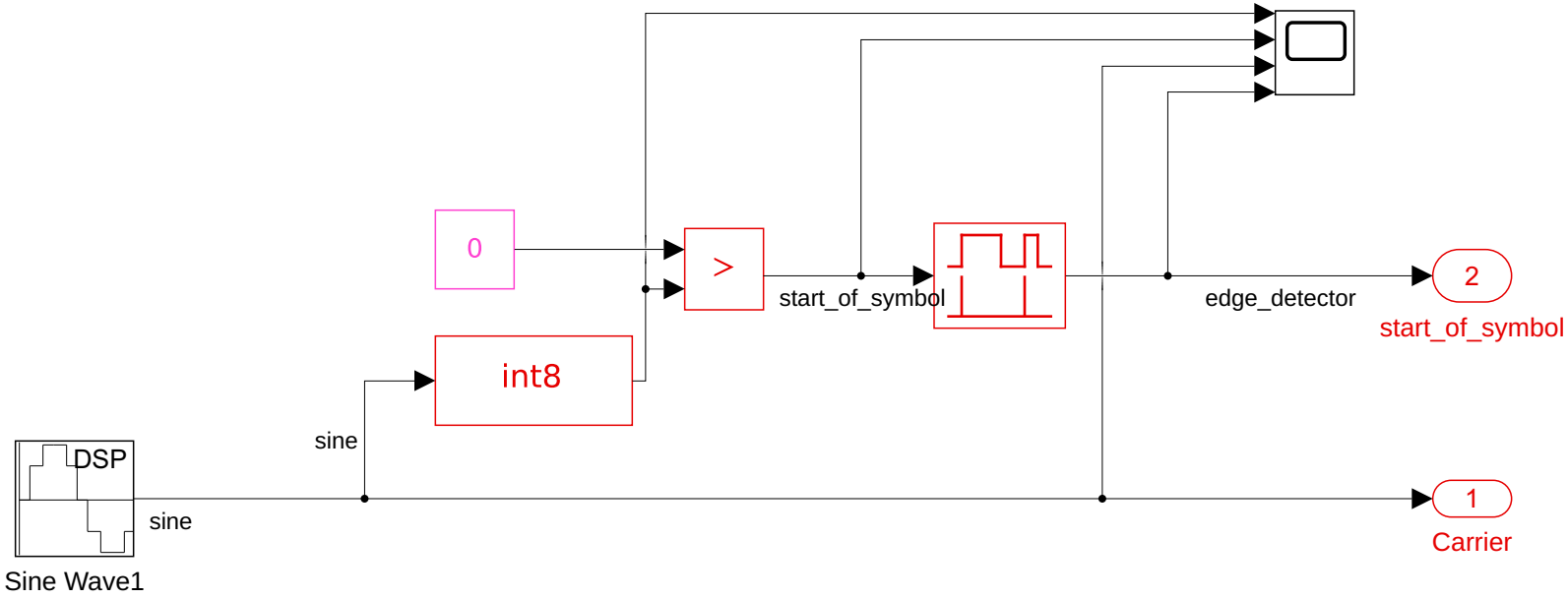

1
Modulated_waveform
Modulated_waveform
i_demod
q_demod
start_of_symbol
Demodulator_with_output_register
2
start_of_symbol
bit0
bit1
convert
Convert_to_decimal
dibit_out
1
dibit_out

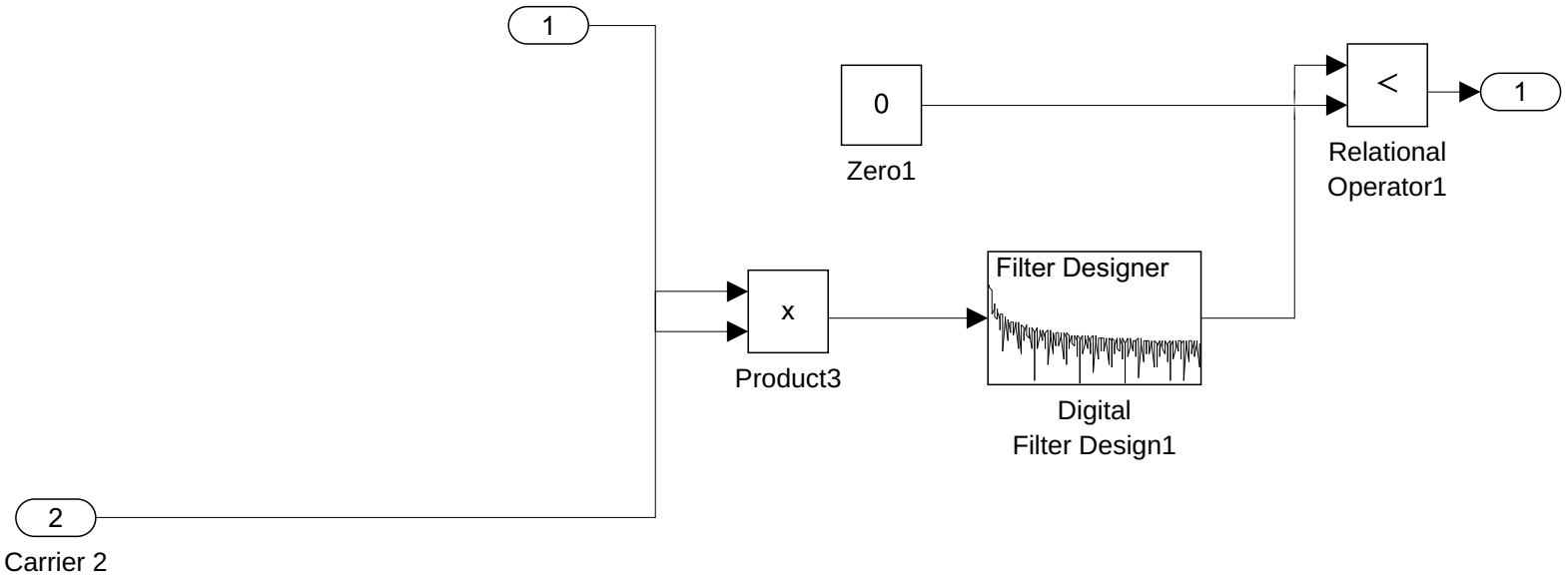

1
bit0
2
2
bit1
+
+
convert
1
convert


1
Modulated_waveform
Modulated_waveform
i_demodulator
q_demodulator
start_of_symbol
Demodulator
i_demod
q_demod
bit_msb
i_data
bit_msb
i_data
i_demod
q_demod
1
i_demod
2
q_demod
3
start_of_symbol

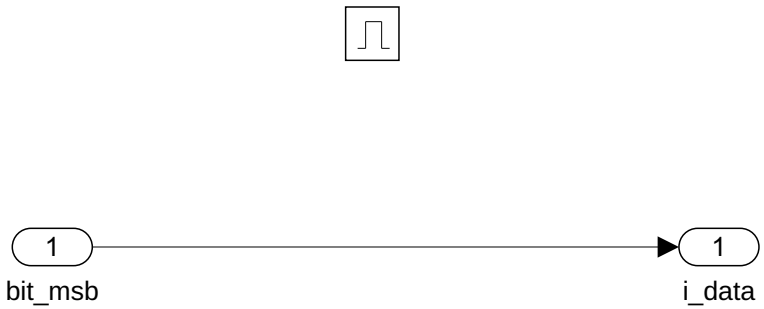

Carrier
carrier
start_of_symbol
I-Phase Carrier Generator
3
start_of_symbol
Carrier
1
2
i_demodulator
1
i_demodulator
I Demodulator
1
Modulated_waveform
1
1
Carrier 2
q_demodulator
2
q_demodulator
Q-Demodulator
Sine Wave2
Carrier 2
Q-Phase Carrier Generatot

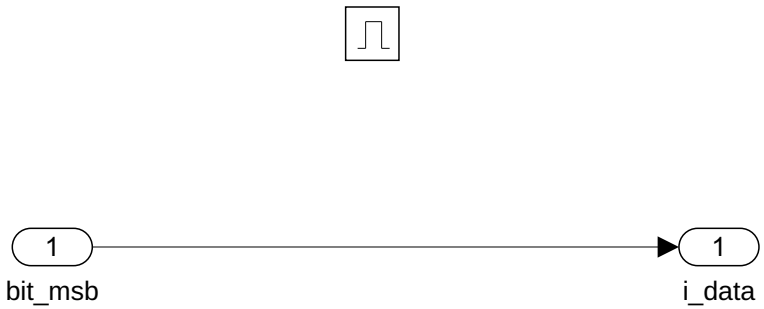

1
Carrier
x
Product2
mixed
Filter Designer
filter_out
Digital
Filter Design
0
Zero
2
<
Relational
Operator
1


0
>
start_of_symbol
edge_detector
2
start_of_symbol
int8
sine
DSP
sine
Sine Wave1
1
Carrier


1
0
Zero1
<
Relational
Operator1
1
Filter Designer
x
Product3
Digital
Filter Design1
2
Carrier 2

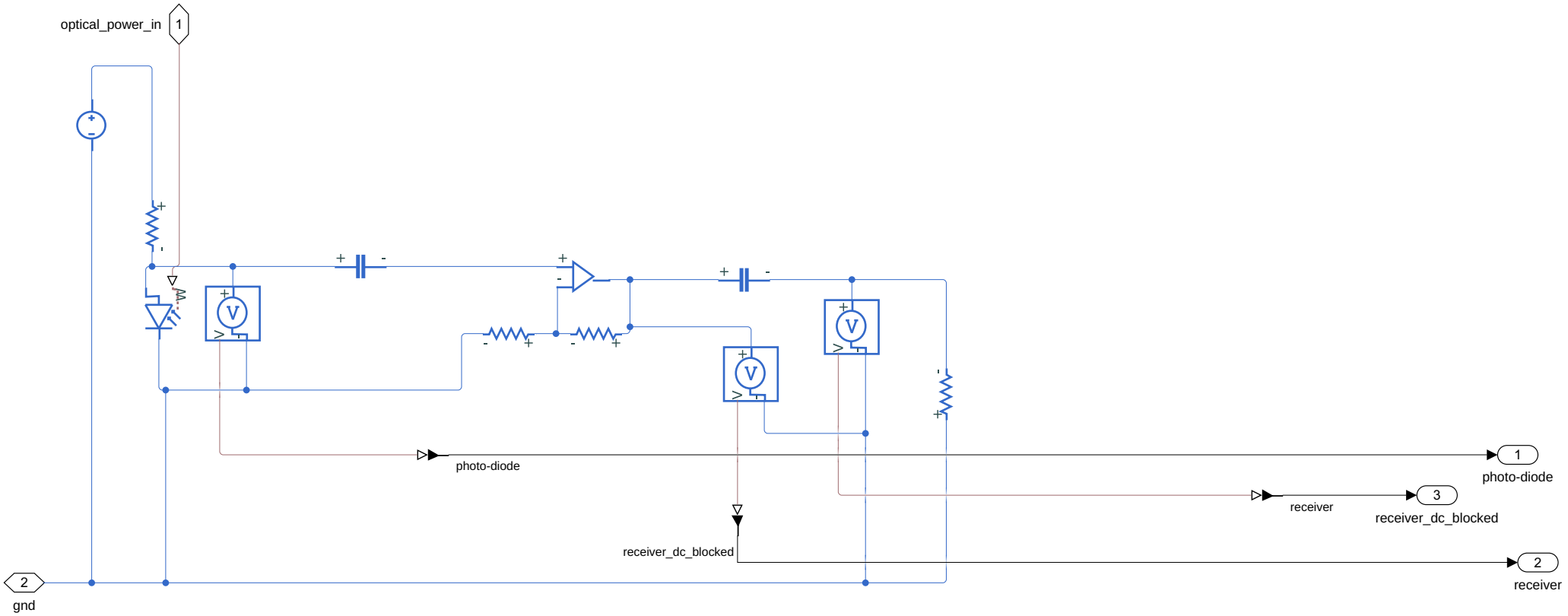

DSP
1
Carrier 2
Sine Wave2

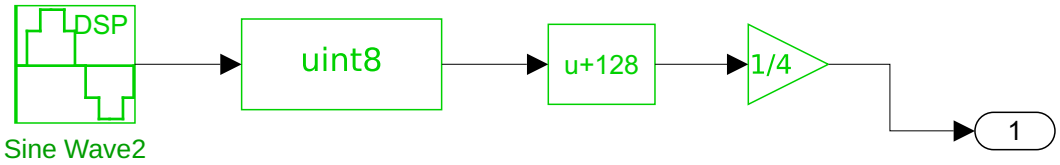

1
bit_msb
1
i_data

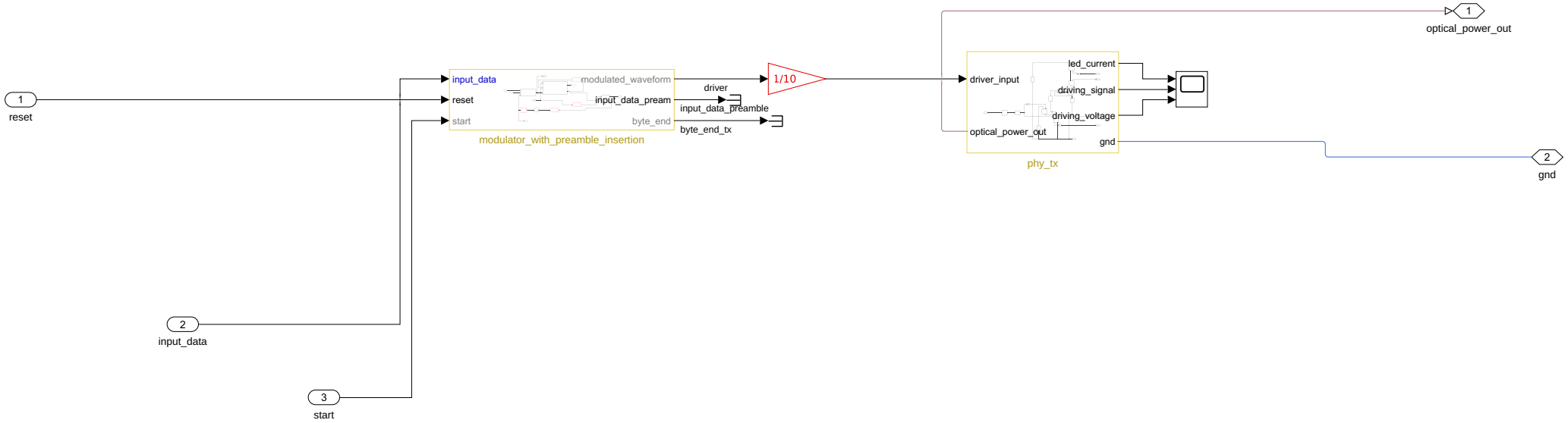

1
bit_msb
1
i_data

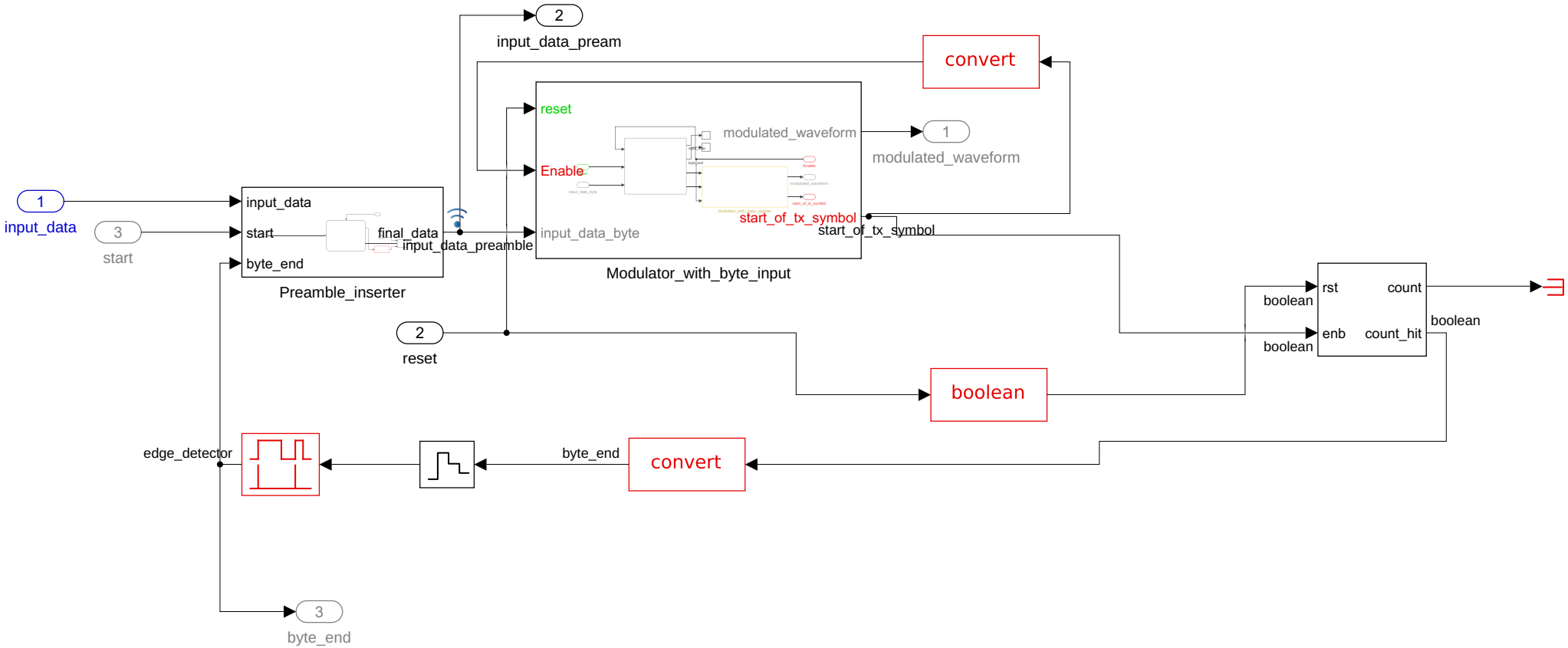
```
function preamble_detected = detect_preamble(dibit, start_of_symbol)
%DETECT_PREAMBLE Detects a run of at least 31 consecutive dibit==2,
%followed by dibit==3 (only evaluated when start_of_symbol == 1).
%
% State Machine:
%   State 0: Counting consecutive dibit==2
%            → If count >= 31, transition to State 1
%   State 1: Stay in this state as long as dibit==2
%            → If dibit==3, trigger detection and reset
%            → Else, reset

    % Persistent state and counter
    persistent state count_two
    if isempty(state)
        state = 0;
        count_two = 0;
    end

    % Default output
    preamble_detected = false;

    if start_of_symbol == 1
        switch state
            case 0
                if dibit == 2
                    count_two = count_two + 1;
                    if count_two >= 31
                        state = 1; % Start monitoring for dibit==3
                    end
                else
                    count_two = 0;
                end

            case 1
                if dibit == 2
                    % Stay in this state, continue accepting more 2s
                    % No change to count_two
                elseif dibit == 3
                    % Preamble matched
                    preamble_detected = true;
                    % Reset
                    state = 0;
                    count_two = 0;
                else
                    % Invalid sequence; reset
                    state = 0;
                    count_two = 0;
                end
        end
    end
end
```

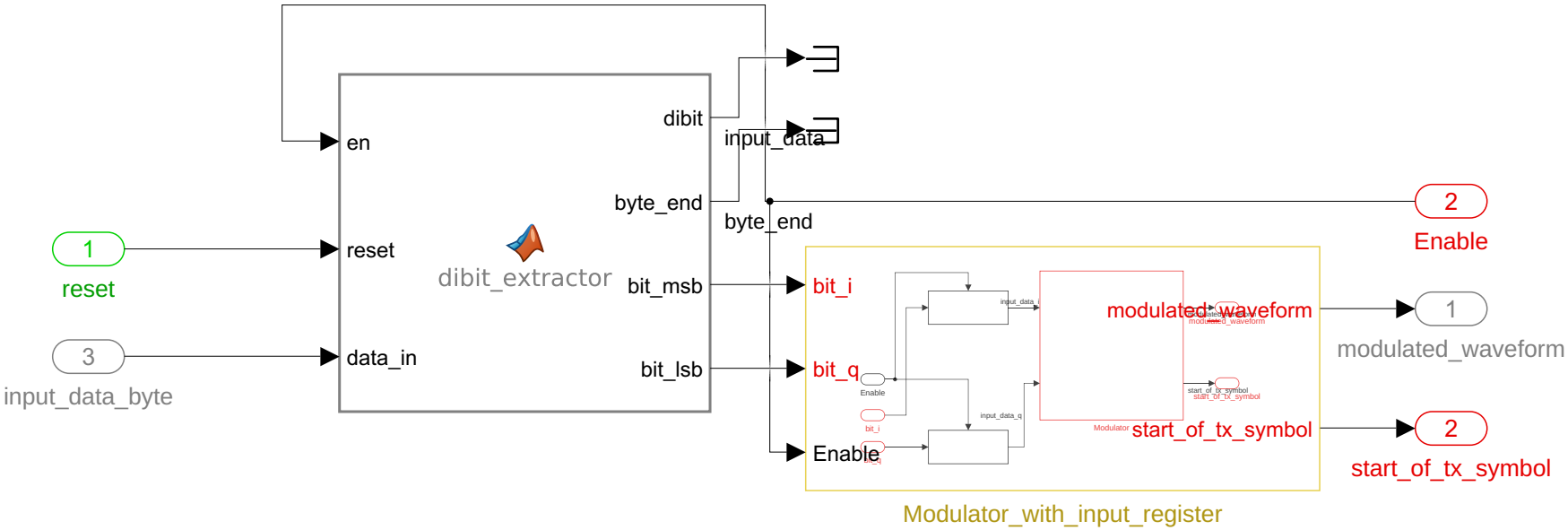
```
function [data_out,byte_end] = byte_assembler(en, reset, dibit)
% byte_assembler  Assembles 8-bit data from four 2-bit segments
%   en        : boolean enable signal (rising edge triggers input sampling)
%   reset     : boolean reset signal (synchronous reset of internal state)
%   dibit     : ufix2 2-bit input
%   data_out  : uint8 8-bit output, valid after 4 dibits

%#codegen

persistent count temp_byte prev_en
if isempty(count)
    count = uint8(0);
    temp_byte = uint8(0);
    prev_en = 0;
    byte_end = 0;
end

% Synchronous reset
if reset
    count = uint8(1);
    temp_byte = uint8(0);
    byte_end = 0;
end

% Detect rising edge of enable
if en==1 && prev_en==0 && reset==0
    shift = bitshift(uint8(dibit), count * 2);  % Place dibit at the correct position
    mask = bitshift(uint8(3), count * 2);       % Create mask for clearing that position
    temp_byte = bitor(bitand(temp_byte, bitcmp(mask, 'uint8')), shift);

    % Increment count and wrap
    count = mod(count + 1, 4);
end
prev_en = en;

% Output the assembled byte only when 4 dibits have been collected
if count == 0 && reset==0
    data_out = temp_byte;
    byte_end = 1;
else
    data_out = uint8(0);
    byte_end = 0;
end
end
```


optical_power_in
1
photo-diode
receiver_dc_blocked
receiver
1
photo-diode
3
receiver_dc_blocked
2
receiver
2
gnd

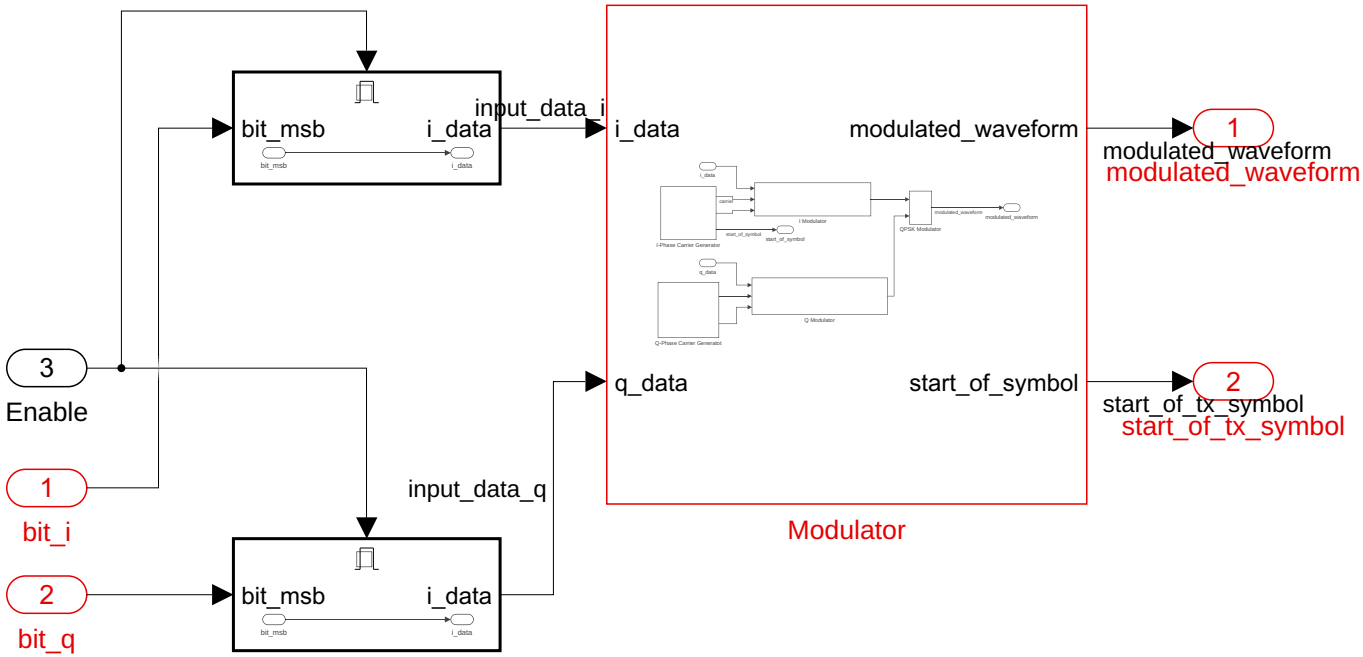

DSP
Sine Wave2
uint8
u+128
1/4
1

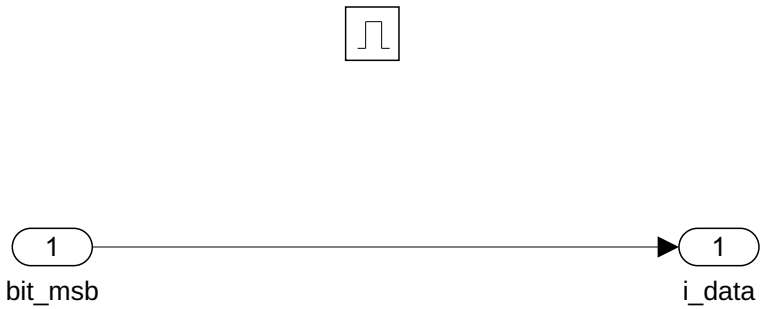

1
reset
2
input_data
3
start
input_data
reset
start
modulated_waveform
input_data_pream
byte_end
modulator_with_preamble_insertion
driver
input_data_preamble
byte_end_tx
1/10
driver_input
optical_power_out
led_current
driving_signal
driving_voltage
gnd
phy_tx
1
optical_power_out
2
gnd

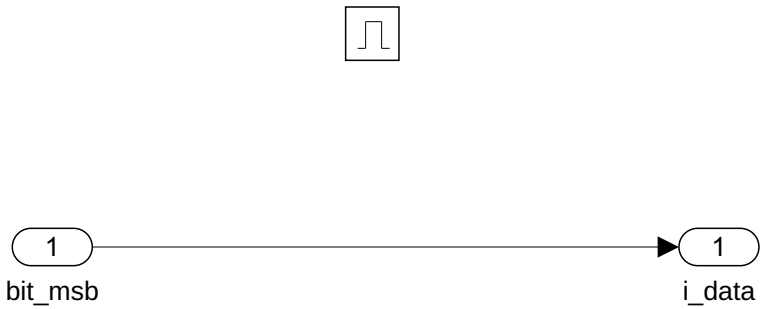

1
input_data
3
start
input_data
start
byte_end
final_data
input_data_preamble
Preamble_inserter
2
input_data_pream
convert
reset
Enable
input_data_byte
modulated_waveform
start_of_tx_symbol
1
modulated_waveform
start_of_tx_symbol
Modulator_with_byte_input
2
reset
boolean
rst
enb
count
count_hit
boolean
boolean
boolean
edge_detector
byte_end
convert
3
byte_end

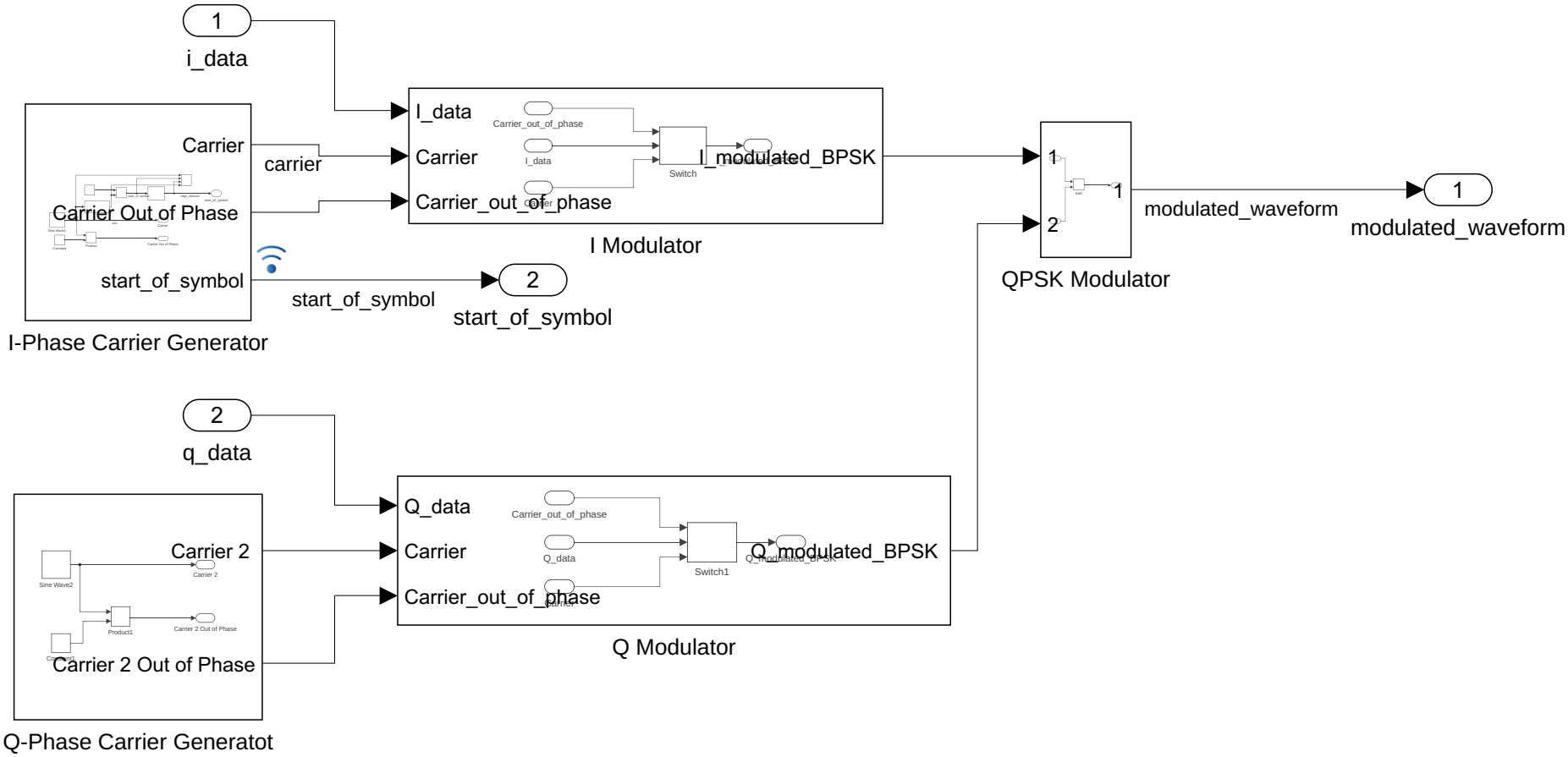

1
reset
3
input_data_byte
en
reset
data_in
dibit_extractor
dibit
byte_end
bit_msb
bit_lsb
input_data
byte_end
bit_i
bit_q
Enable
modulated_waveform
start_of_tx_symbol
2
Enable
1
modulated_waveform
2
start_of_tx_symbol
Modulator_with_input_register

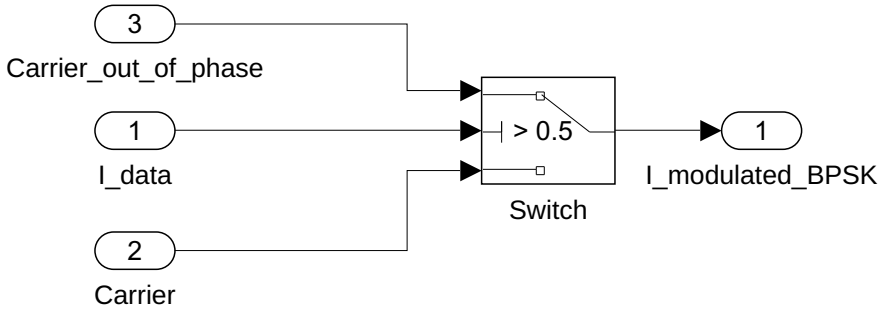
```
function [dibit, byte_end, bit_msb, bit_lsb] = dibit_extractor(en, reset, data_in)
% dibit_extractor  Extracts two-bit segments from an 8-bit input on enable pulses
%   en        : boolean enable signal (rising edge triggers output update)
%   reset     : boolean reset signal (synchronous reset of internal state)
%   data_in   : uint8 8-bit input data
%   dibit     : UFix_2_0 2-bit unsigned fixed-point output
%   byte_end  : real scalar; goes high every four dibits (i.e., end of byte)
%   bit_msb   : logical scalar; the most-significant bit of the dibit
%   bit_lsb   : logical scalar; the least-significant bit of the dibit

%#codegen

persistent count prev_en
if isempty(count)
    count   = uint8(3);   % cycle index (0 to 3)
    prev_en = 0;          % previous enable state
end

% Default outputs
byte_end = 0;
bit_msb  = false;
bit_lsb  = false;

% Synchronous reset
if reset
    count = uint8(0);
else
    % Detect rising edge of enable
    if en == 1 && prev_en == 0
        % Increment and wrap count
        count = mod(count + 1, 4);

        % Pulse byte_end when completing the fourth dibit
        if count == 1
            byte_end = 1;
        end
    end
end

prev_en = en;

% Determine starting bit position (0-based)
startBit = count * 2;

% Extract two bits (MATLAB bitget uses 1-based positions)
bit0 = bitget(data_in, startBit + 1);  % LSB of the dibit
bit1 = bitget(data_in, startBit + 2);  % MSB of the dibit

% Expose individual bits as Booleans
bit_lsb = logical(bit0);
bit_msb = logical(bit1);

% Assemble value (bit1 is MSB, bit0 is LSB)
value = uint8(bit1) * 2 + uint8(bit0);

% Create 2-bit unsigned fixed-point output
dibit = fi(value, 0, 2, 0);
end
```

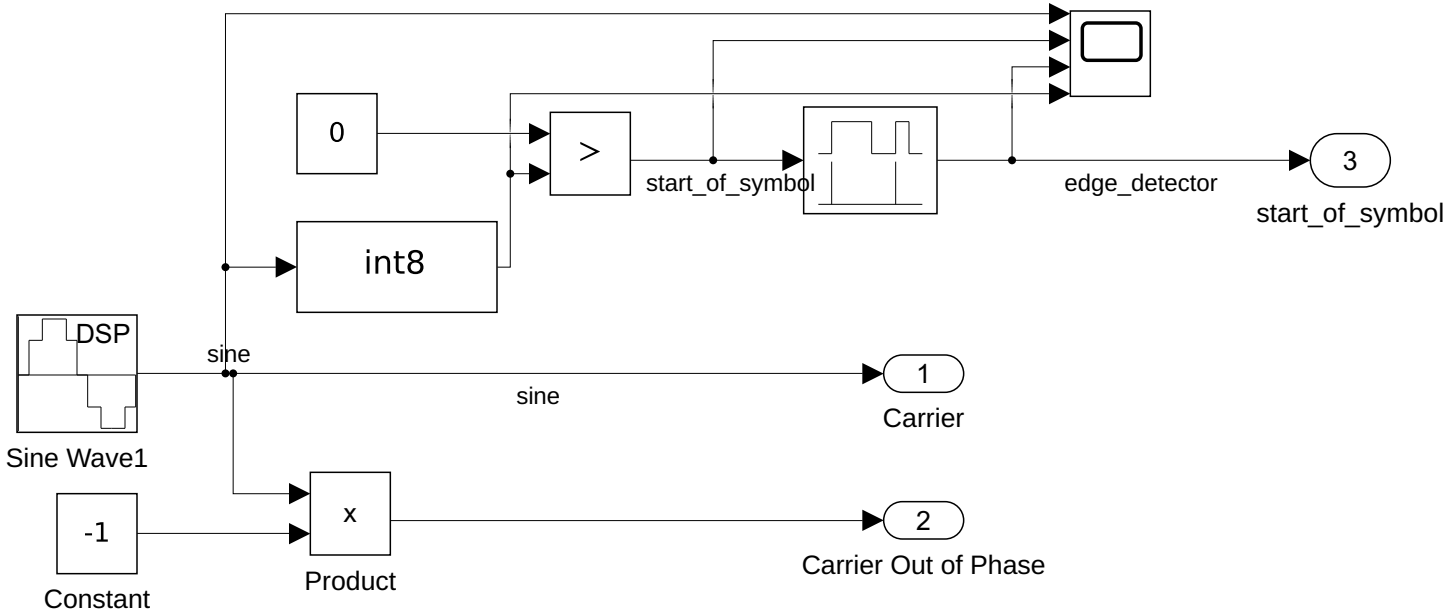

3
Enable
1
bit_i
2
bit_q
bit_msb
i_data
input_data_i
input_data_q
i_data
q_data
modulated_waveform
start_of_symbol
Modulator
1
modulated_waveform
modulated_waveform
2
start_of_tx_symbol
start_of_tx_symbol

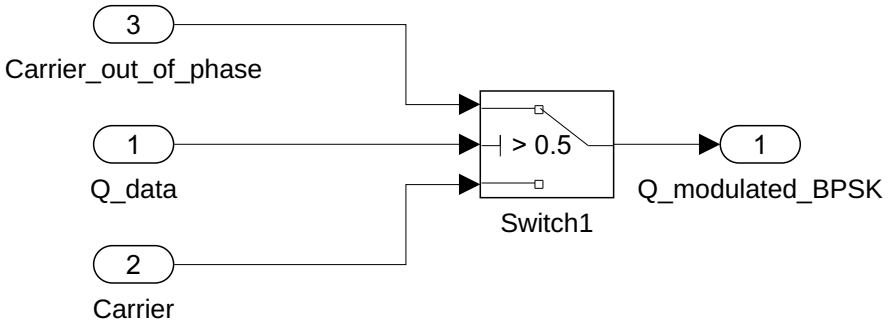

1
bit_msb
1
i_data

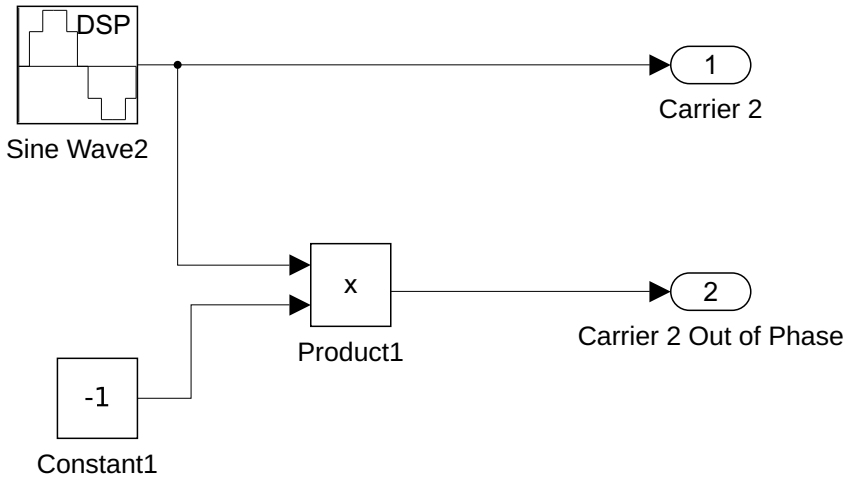

1
bit_msb
1
i_data

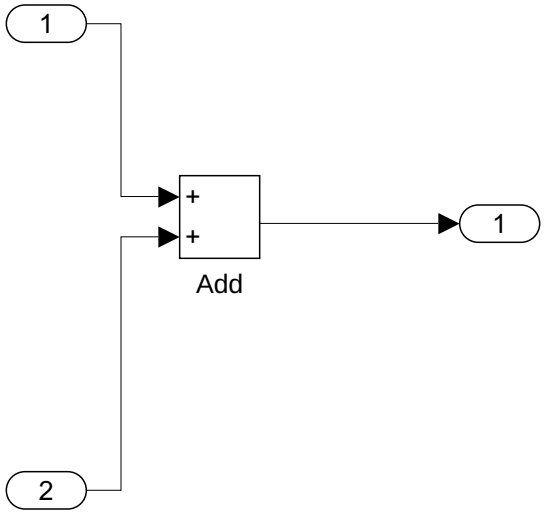

1
i_data
Carrier
carrier
Carrier Out of Phase
start_of_symbol
I-Phase Carrier Generator
start_of_symbol
2
start_of_symbol
I_data
Carrier
Carrier_out_of_phase
Carrier_out_of_phase
I_data
Switch
I_modulated_BPSK
I Modulator
1
2
1
QPSK Modulator
modulated_waveform
1
modulated_waveform
2
q_data
Carrier 2
Carrier 2 Out of Phase
Q-Phase Carrier Generatot
Q_data
Carrier
Carrier_out_of_phase
Carrier_out_of_phase
Q_data
Switch1
Q_modulated_BPSK
Q Modulator

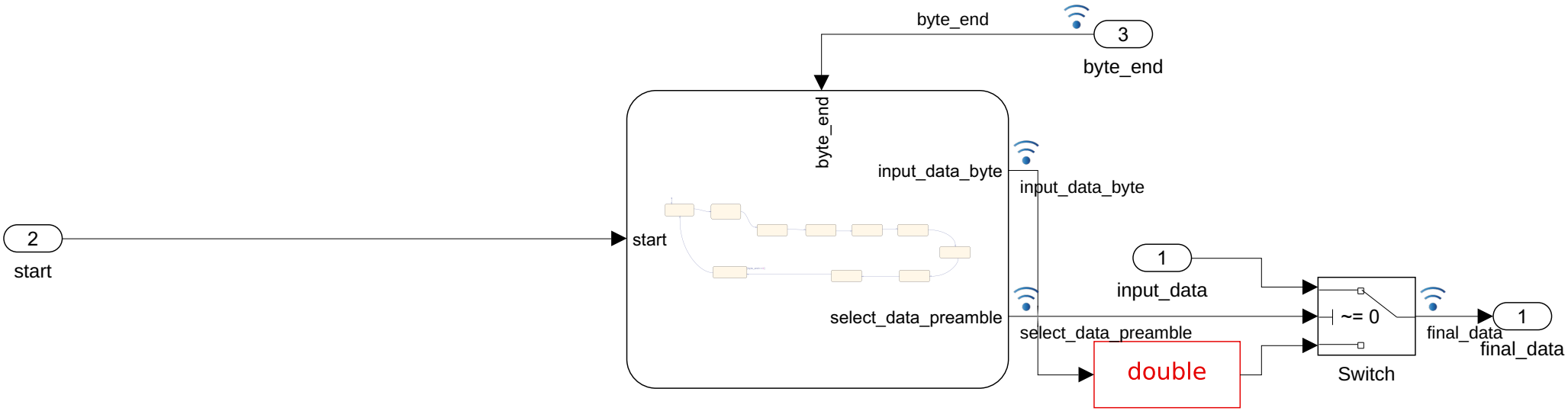

3
Carrier_out_of_phase
1
I_data
2
Carrier
> 0.5
Switch
1
I_modulated_BPSK

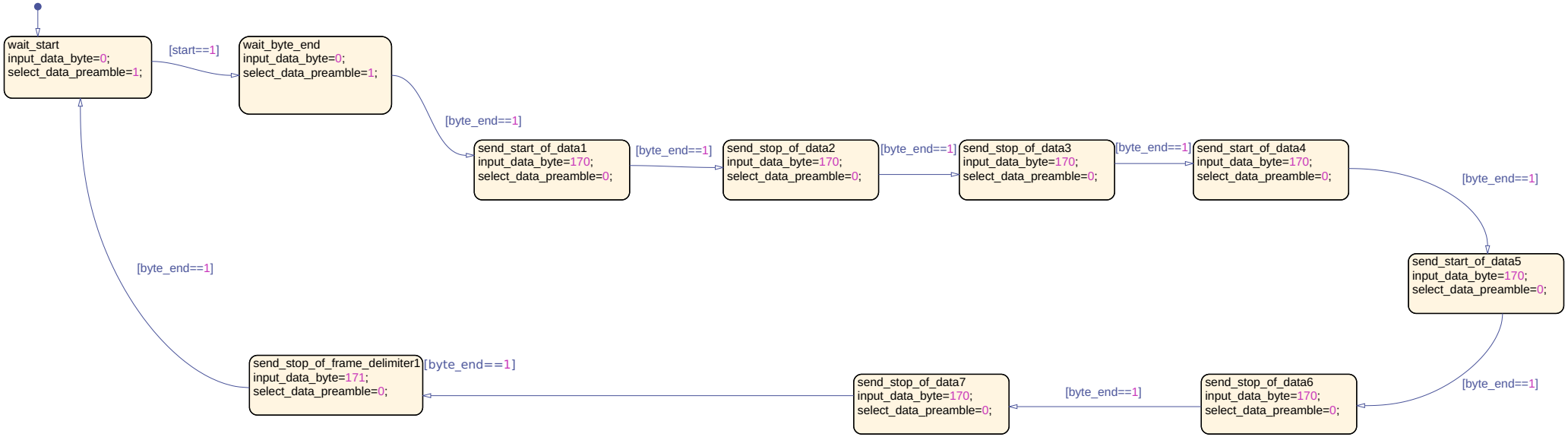

0
>
start_of_symbol
edge_detector
3
start_of_symbol
int8
DSP
sine
sine
1
Carrier
Sine Wave1
-1
x
2
Carrier Out of Phase
Constant
Product

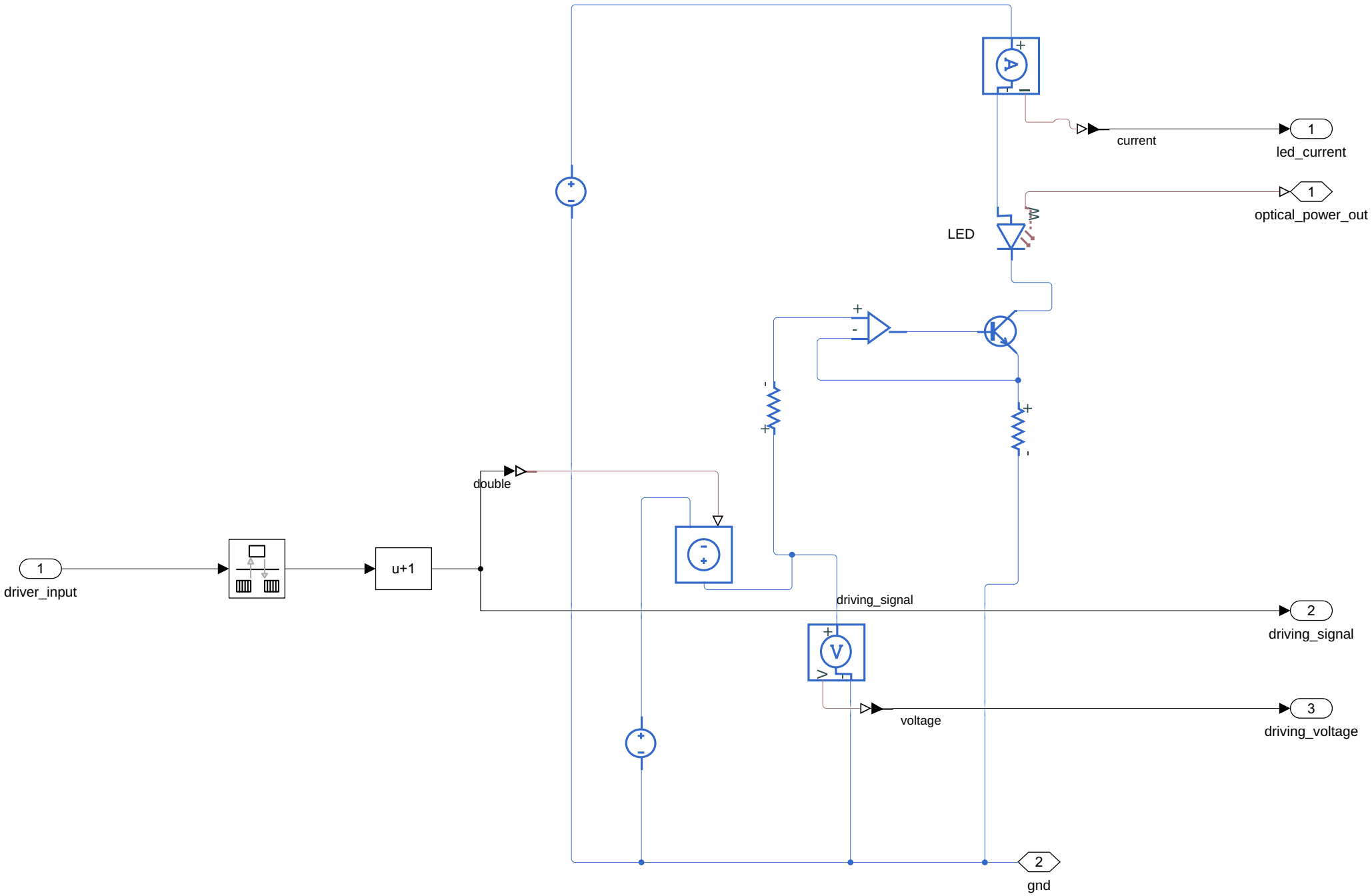

3
Carrier_out_of_phase
1
Q_data
2
Carrier
> 0.5
Switch1
1
Q_modulated_BPSK


DSP
Sine Wave2
1
Carrier 2
x
Product1
2
Carrier 2 Out of Phase
-1
Constant1


1
+
+
Add
1
2


byte_end
3
byte_end
byte_end
input_data_byte
input_data_byte
2
start
start
1
input_data
select_data_preamble
select_data_preamble
~= 0
final_data
1
final_data
double
Switch


wait_start
input_data_byte=0;
select_data_preamble=1;
[start==1]
wait_byte_end
input_data_byte=0;
select_data_preamble=1;
[byte_end==1]
send_start_of_data1
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_stop_of_data2
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_stop_of_data3
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_start_of_data4
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_start_of_data5
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_stop_of_data6
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_stop_of_data7
input_data_byte=170;
select_data_preamble=0;
[byte_end==1]
send_stop_of_frame_delimiter1
input_data_byte=171;
select_data_preamble=0;
[byte_end==1]


1
driver_input
u+1
double
LED
current
1
led_current
1
optical_power_out
driving_signal
2
driving_signal
voltage
3
driving_voltage
2
gnd