Carleton UNIVERSITY Department of Systems and Computer Engineering

SYSC 5104 METHODOLOGIES FOR DISCRETE EVENT MODELLING AND SIMULATION

Assignment #1

Part #2

Communications Management System (CMS)

Student # 100868021

Colin Timmons

29 October 2011

TABLE OF CONTENTS

1. BACKGROUND 4

2. MODEL BASIS 5

3. CMS COUPLED MODEL DEVS FORMALISM 7

3.1 CMS TESTING STRATEGY 8
3.1.1 Requirement 1 8
3.1.2 Requirements 2-9 9
3.1.3 Requirement 10 11
3.1.4 Requirement 11 11

CSU COUPLED MODEL DEVS FORMALISM 13

4.1 CSU TESTING. 14
4.2 CSU INTERFACE CONTROLLER 15
4.2.1 CSU InterfaceTesting. 18
4.3 BUTTON CONTROLLER 19
4.3.1 Button Controller Testing 21
4.4 CRYPTO CONTROLLER 22
4.4.1 Crypto Controller Testing 22
4.5 GUI CONTROLLER 23
4.5.1 GUI Controller Testing. 26
4.6 LED 27
4.6.1 LED Testing. 27

5 HF COUPLED MODEL DEVS FORMALISM 29

5.1 HF TESTING 29
5.2 ANVDT 30
5.3.1 ANVDT Testing 32
5.3 HF_TRANSCEIVER 32
5.3.1 HF_Transceiver Testing 34

6 CCU COUPLED MODEL DEVS FORMALISM 35

6.1 CCU TESTING 37
6.2 CCU INTERFACE 39
6.2.1 CCU Interface Testing 41
6.3 STATUS CONTROLLER 42
6.4.1 Status Controller Testing 44
6.5 AUDIO MULTIPLEXER 44
6.5.1 Audio Multiplexer Testing 47

7. EXCLUSIONS 48

7.2 VUHF RECEIVER-TRANSMITTER 48
7.3 AM RECEIVER-TRANSMITTER 49
7.4 FM RECEIVER-TRANSMITTER 49
7.5 LED 49

7.6 CSU 49
7.7 CCU 49
7.8 RADIOS 49

8 STATE DIAGRAMS 50

9 RESULTS 52

TABLE OF FIGURES

Figure 1. CMS Coupled Model 4
Figure 2. About CMS Simulator Dialog Box 5
Figure 3. CMS - CSU Navcom Dialog. One of 12 possible stations. 5
Figure 4. CMS - HF Simulator Dialog 6
Figure 5. CMS - CCU Simulator 6
Figure 7. CSU Coupled Model 13
Figure 8. HF Coupled Model 29
Figure 9. CCU Coupled Model 35
Figure 10. CSU State Diagram - CSU User Selection 50
Figure 11. Crypto State Diagram - Crypto User Selection 50
Figure 12. CMS Transmission Radio Activity State Diagram 51
Figure 13. CMS Reception Radio Activity State Diagram 51
Figure 14. Atomic Modeling of the CSU Operation 53
Figure 15. Atomic Modeling of the CCU Operation 54
Figure 16. Atomic Modeling of the HF Operation 55

Tables

Table 1. Requirement 1 Testing 9
Table 2. Requirements 2-9 Testing 10
Table 3. Requirement 10 Testing 11
Table 4. Requirement 11 Testing 12
Table 5. CSU Testing 14
Table 6. CSU Interface Testing 18
Table 7. Button Controller Testing 22
Table 8. Crypto Controller Testing 23
Table 9. GUI Controller Testing 26
Table 10. LED Testing 28
Table 11. HF Testing 30
Table 12. ANVST Testing 32
Table 13. HF Transceiver Testing 34
Table 14. CCU Testing 39
Table 15. CCU Interface Testing 42
Table 16. Status Controller Testing 44
Table 17. Audio Multiplexer Testing 48

1. Background

The conceptual model for this assignment is a Communications Management System (CMS) that is used on the CP-140 Aurora aircraft. The purpose of this model to ensure the voice operation of the simulator matches the hardware implementation as represented on the aircraft.

The CMS allow the crew to communicate internally and externally with other platforms. The CMS is composed of 12 Crew Station Units (CSU), dual redundant Communication Control Units (CCU), four secure Very Ultra High Frequency radios (VUHF), a Amplitude Modulated (AM) radio, a Frequency Modulated (FM) radio, two High Frequency (HF) radios, a secure Advanced Narrowband Secure Voice Terminal (ANVDT), 12 keyline headset and footswitches.

For the purpose of this model, the CMS will be reduced to 1 CSU, 1 CCU, 1 keyline switch, 1 HF radio, and 1 VUHF radio. The many configuration selections of the CSU will be neglected and focus on the VUHF transmission in secure and non-secure mode. Similarly only voice message transmissions will be used while the various data or sonobouy message transmissions will be neglected. Because of the security implications for the crypto, these components will not be decomposed.

Conceptually the CMS model appears as follows:

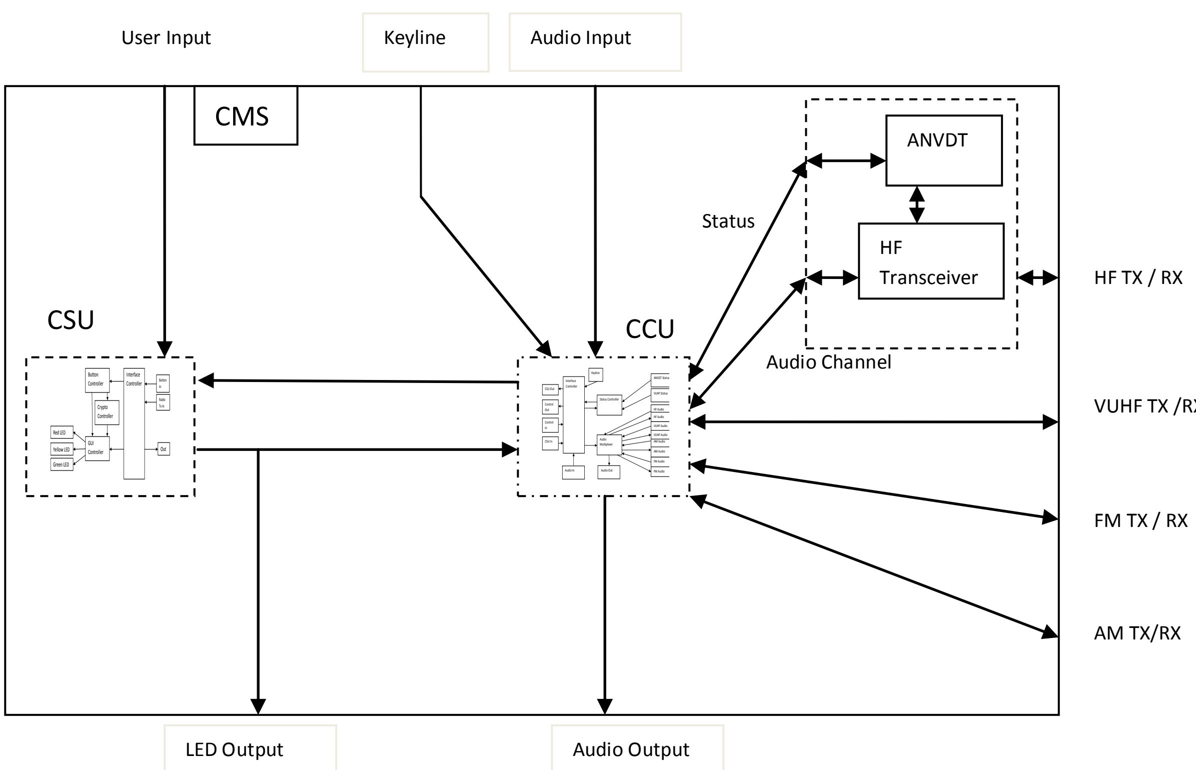

Figure 1. CMS Coupled Model

The CMS behaviour is such that it must handle voice activation key from the CCU, handle user input button selection for a VUHF radio control head, handle user input crypto button selection for secure transmission selection, toggle led states on the CSU to indicate radio operation for listening, transmission and secure operations, transmit radio state parameters to CCU, handle single click and double click button selection on the CSU, initiate key line activation of radio control head; and remember last crypto setting for each radio.

2. Model Basis

I have presented a screen capture of the actual software application that I created to help aid in the understanding.

Figure 2. About CMS Simulator Dialog Box

Figure 3. CMS - CSU Navcom Dialog. One of 12 possible stations.

Figure 4. CMS - HF Simulator Dialog

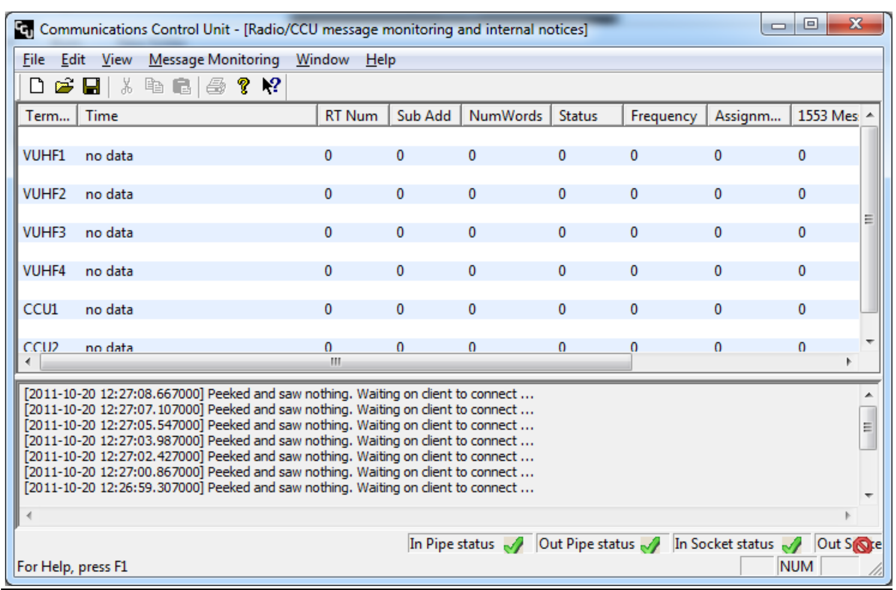

Figure 5. CMS - CCU Simulator

3. CMS Coupled Model DEVS Formalism

C = < X, Y, D, { M_d | d ϵ D }, EIC, EOC, IC, select >

where X = { ((Keyline, v) | Keyline ∈ IPorts, v ∈ N),

((Audio_Input, v) | Audio_Input ∈ IPorts, v ∈ N),

((User_Input, v) | User_Input ∈ IPorts, v ∈ N),

((HF_Reception_In, v) | HF_Reception_In ∈ IPorts, v ∈ N),

((VUHF_Reception_In, v) | VUHF_Reception_In ∈ IPorts, v ∈ N),

((AM_Reception_In, v) | AM_Reception_In ∈ IPorts, v ∈ N),

((FM_Reception_In, v) | FM_Reception_In ∈ IPorts, v ∈ N) };

Y = { ((Audio_Speaker_Out, v) | Audio_Speaker_Out ∈ OPorts, v ∈ N),

((HF_Transmission_Out, v) | HF_Transmission_Out ∈ OPorts, v ∈ N),

((VUHF_Transmission_Out, v) | VUHF_Transmission_Out ∈ OPorts, v ∈ N),

((AM_Transmission_Out, v) | AM_Transmission_Out ∈ OPorts, v ∈ N),

((FM_Transmission_Out, v) | FM_Transmission_Out ∈ OPorts, v ∈ N),

((LED_Output, v) | LED_Output ∈ OPorts, v ∈ N) };

D = { CSU, CCU, HF, VUHF_Transceiver, AM_Transceiver, FM_Transceiver };

M_d = { M_{CSU}, M_{CCU}, M_{HF}, $M_{VUHF\_Transceiver}$, $M_{AM\_Transceiver}$, $M_{FM\_Transceiver}$ };

EIC ⊆ {((Self, Keyline), (CCU, Keyline)),
((Self, Audio_In), (CCU, Audio_In)),
((Self, User_Input), (CSU, User_Input)),
((Self, HF_Reception_In), (HF, HF_Reception_In)),
((Self, VUHF_Reception_In), (CCU, VUHF_Audio_In)),
((Self, FM_Reception_In), (CCU, FM_Audio_In)),
((Self, AM_Reception_In), (CCU, AM_Audio_In)) };

EOC ⊆ { ((CSU, Status_Out), (CCU, CSU_In)),
((CCU, Audio_Out), ((Self, Audio_Speaker_Out)),
((HF_Transmission_Out@HF Self, HF_Transmission_Out)),
((CSU, Status_Out), (Self, LED_Output)),
((CCU, AM_Audio_Out), (Self, AM_Transmission_Out)),
((CCU, FM_Audio_Ou), (Self, FM_Transmission_Out)),
((CCU, VUHF_Audio_Out), (Self, VUHF_Transmission_Out)) }

IC ⊆ { ((HF, Hf_Audio_Out), (CCU,HF_Audio_In)),
((HF, ANVDT_Status_Out), (CCU, ANVDT_Status_In)),
((CCU, HF_Audio_Out), (HF, HF_Audio_In)),
((CCU, CSU_Out@CCU CSU, CSU, Radio_Tx_In)),
((CCU, ANVDT_Status_Out), (HF, HF_Status_In)) }

3.1 CMS Testing Strategy

Obviously, as any black and white box testing, there basis on the testing is determined by the requirements. In this case, the requirements are a simplification of the actual operating parameters excluding bitmap map manipulation, overloading and creating new visual objects, the requirement of non-activeX components, auditory waveform manipulation and the whole aspect of command and control of the Mil-Std-1553B architecture through the Cockpit Display Units (CDUs). As this greatly simplifies matters the DEVs CMS, the requirements for black box testing of the CMS are as follows:

1. Handle voice activation key to the Communication Control Unit (CCU);
2. Handle user input button selection for an individual Very Ultra High Frequency (VUHF) radio control head;
3. Handle user input button selection for an individual High Frequency (HF) radio control head;
4. Handle user input button selection for an individual Frequency Modulated (FM) radio control head;
5. Handle user input button selection for an individual Amplitude Modulated (AM) radio control head;
6. Handle crypto input button selection for crypto selection;
7. Toggle led states:
    a. Yellow – listening mode,
    b. Red – transmission mode, and
    c. Green – crypto mode;
8. Handle single click and double click button selection;
9. Remember last crypto setting for each VUHF radio.
10. Transmit radio state parameters to CCU;
11. Initiate key line activation of radio control head; and

Based on these limited requirements the following overall testing strategy can be put into place.

3.1.1 Requirement 1

Activate a standalone keyline input into the CMS coupled model. As this should have the effect of an open microphone if a radio was selected. However, with no radio selection inputted into the CSU, there should be no output.

Hence, the input event file should have an entry as follows:

Input	Remarks	Output
00:00:00:00 Keyline 28	Keyed microphone switch	0
00:00:01:00 Keyline 29	UnKeyed microphone switch	0
00:00:02:00 Keyline 28	Keyed microphone switch	0
00:00:03:00 Keyline 29	UnKeyed microphone switch	0

Table 1. Requirement 1 Testing

Result: The keyline activation should be accepted by the CCU and its CCU interface should send then examine whether the CSU has a valid radio configuration, the CCU Status Controller should determine that a radio is present and operational, and that a valid keyline is present. Without all three present the CCU Interface should refrain from outputting any messages. The values given are range checked to ensure that false data is not activating a live transmission nor providing auditory stimulation.

3.1.2 Requirements 2-9

These requirements can be satisfied simultaneously. The CMS should handle three different inputs. The first would be a single press event on one of the radio selections on the CSU. For this model, the user input would be for the HF, VUHF, AM or FM radio as described in the Declarations.h file. Determination of which radio is selected is done by bit shifting or masking the event file input for the radio selected and the user input action. For example 0x031 denotes the HF radio and a single click event occurrence. 0x332 indicates the FM radio and a double click event input.

Thus the event file input for this requirement would be a single press, single press, single press, single press, crypto press, single press, a double press for each radio and a single press to restart the sequence to ensure coverage by turning it on. The event file input would be similar to the following where the last number is the decimal conversion for the hexadecimal numbers and the selected radio:

Input	Remarks	Output Led_Output port (Hexadecimal		Decimal)
00:00:00:00 User_Input 49	Yellow HF LED	0x00010001	65537
00:00:01:00 User_Input 49	Red HF LED	0x00010011	65553
00:00:02:00 User_Input 49	Yellow HF LED	0x00010001	65537
00:00:03:00 User_Input 49	Red HF LED	0x00010011	65553
00:00:04:00 User_Input 51	Crypto mode Green HF LED	0x00010111	65809
00:00:05:00 User_Input 49	Yellow HF LED (secure)	0x00010101	65793
00:00:06:00 User_Input 50	HF Off	0x00010100	65792
00:00:07:00 User_Input 49	Yellow HF LED	0x00010101	65793
00:00:08:00 User_Input 49	Red HF LED	0x00010111	65809
00:00:08:00 User_Input 51	Crypto mode Green HF LED Off	0x00010011	65553
00:00:09:00 User_Input 49	Yellow HF LED	0x00010001	65537
00:00:10:00 User_Input 49	Red HF LED	0x00010011	65553
00:00:11:00 User_Input 50	HF Off	0x00010000	65536
00:00:20:00 User_Input 305	Yellow VUHF LED	0x00100001	1048577
00:00:21:00 User_Input 305	Red VUHF LED	0x00100011	1048593
00:00:22:00 User_Input 305	Yellow VUHF LED	0x00100001	1048577

00:00:23:00 User_Input 305	Red VUHF LED	0x00100011	1048593
00:00:24:00 User_Input 307	Crypto mode Green VUHF LED	0x00100111	1048849
00:00:25:00 User_Input 305	Yellow VUHF LED	0x00100111	1048833
00:00:26:00 User_Input 306	VUHF Off	0x00100100	1048832
00:00:27:00 User_Input 305	Yellow VUHF LED	0x00100101	1048833
00:00:28:00 User_Input 305	Red VUHF LED	0x00100111	1048849
00:00:29:00 User_Input 307	Crypto mode Green VUHF LED Off	0x00100011	1048593
00:00:30:00 User_Input 305	Yellow VUHF LED	0x00100001	1048577
00:00:31:00 User_Input 305	Red VUHF LED	0x00100011	1048593
00:00:32:00 User_Input 306	VUHF Off	0x00100000	1048576
00:00:40:00 User_Input 561	Yellow FM LED	0x01000001	16777217
00:00:41:00 User_Input 561	Red FM LED	0x01000011	16777233
00:00:42:00 User_Input 561	Yellow FM LED	0x01000001	16777217
00:00:43:00 User_Input 561	Red FM LED	0x01000011	16777233
00:00:45:00 User_Input 561	Yellow FM LED	0x01000111	16777489
00:00:46:00 User_Input 562	FM Off	0x01000100	16777472
00:00:47:00 User_Input 561	Yellow FM LED	0x01000101	16777473
00:00:48:00 User_Input 561	Red FM LED	0x01000011	16777233
00:00:50:00 User_Input 561	Yellow FM LED	0x01000001	16777217
00:00:51:00 User_Input 561	Red FM LED	0x01000011	16777233
00:00:52:00 User_Input 562	FM Off	0x01000000	16777216
00:01:00:00 User_Input 817	Yellow AM LED	0x10000001	268435457
00:01:01:00 User_Input 817	Red AM LED	0x10000011	268435473
00:01:02:00 User_Input 817	Yellow AM LED	0x10000001	268435457
00:01:03:00 User_Input 817	Red AM LED	0x10000011	268435473
00:01:05:00 User_Input 817	Yellow AM LED	0x10000111	268435729
00:01:06:00 User_Input 818	AM Off	0x10000100	268435712
00:01:07:00 User_Input 817	Yellow AM LED	0x10000101	268435713
00:01:08:00 User_Input 817	Red AM LED	0x10000011	268435473
00:01:09:00 User_Input 817	Yellow AM LED	0x10000001	268435457
00:01:10:00 User_Input 817	Red AM LED	0x10000011	268435473
00:01:11:00 User_Input 818	AM Off	0x10000000	268435456

Table 2. Requirements 2-9 Testing

If radio is selected, the CSU should light up the led in an off to yellow to red to yellow sequence. If a crypto selection is made the CSU will activate green only after the first transition to a red (transmitting) state and then remain in secure mode until the radio is either deselected from secure mode or the radio is turned off. As the Mil-Std-1553B bus is naturally not being simulated here and the CDUs are the controller for the radio power state, the DEVs simulator will only go from secure to unsecure for only the HF and VUHF radios through the input toggling of the crypto input as shown above.

3.1.3 Requirement 10

The radios will transmit with their status to the CCU identifying whether they are operational. This is performed by the CCU status controller who naturally would not send a radio transmission unless the radio is operational. Similarly, the CCU would not receive a transmission from an outside agency unless the radio was operational. The CCU Interface upon an input would inform the Status Controller that an input was received and to poll the radios for their status upon each instance. This is similarly to a Built-In Test (BIT) to ensure there is no damage when the Radio Frequency is applied to the couplers. If the radio is reported off-line is remains off-line by the CDU until re-initialization.

The HF is a coupled model of two atomic models, the ANVDT and HF Transceiver. Using a random number generator, the ANVDT reports its status if the random number generated is between 0 and 0.75. Similarly, the number generator is used to valid a presence for the AM transceiver, FM transceiver and the VUHF transceiver. The external transmission will not be received if the status generated by the random number is below the threshold. This is similar to the effects of the aircraft flying and manoeuvring and the effects of the earth and sun on HF frequencies and line of sight communications in rugged combat scenarios.

Thus the testing event file would activate each radio as a transmission.

Input	Remarks	Output (assuming CSU activated)	
		Led_Output	Audio_out
00:00:01:00 HF_Reception_In 40	Reception on the HF	0x00011001	34
00:00:02:00 VUHF_Reception_In 40	Reception on the VUHF	0x00101001	34
00:00:03:00 FM_Reception_In 40	Reception on the FM	0x01001001	34
00:00:04:00 AM_Reception_In 40	Reception on the AM	0x10001001	34

Table 3. Requirement 10 Testing

If the CSU is activated as a yellow LED, the LED would respond as flashing that a radio reception is in progress giving the crew an indication of a radio message. Since the DEVs CSU does not perform any bitmap manipulation, the “flashing” of the LED routed through the CSU as a status message into the CCU. An active Led is not zero and therefore in operation. Thus for each radio LED there will be an output of the Audio_Out indicating success and a byte representation of the radios in the upper word and the lower word would contain the results of the user input on the CSU. A zero value of the audio output indicates that there was no radio reception, no user input, nor keyline.

3.1.4 Requirement 11

Requirement 11 is by far the most difficult in that it requires the capability to handle the transmission in coordination with the CSU, CCU and the radio control heads. In order to transmit voice, the model must take into consideration the user selection, the radio status, the keyline, the audio input and whether the voice is to go in secure mode for the VUHF and HF control heads. As such, it is not a discrete input, that can be broken down but rather relies on the feedback and fidelity of the system as a whole.

Thus the testing can be performed as follows:

Input	Remarks	Output		
		Led_Output (Hexadeciaml – Decimal)	Active Radio Output	Audio_out
00:00:01:00 User_Input 49	Yellow HF LED	0x00010001 - 65537		
00:00:02:00 User_Input 49	Red HF LED	0x00010011 - 65553		
00:00:03:00 Keyline 28	Key the mic	0x00011011 - 69649		
00:00:04:00 Audio_In 1	Audio to be transmitted	0x00011011 - 69649	HF_Transmission_Out 34	34
00:00:05:00 Keyline 29	Unkey the mic	0x00100111 - 1048849		
00:00:06:00 User_Input 305	Yellow VUHF LED	0x00110001 - 1114113		
00:00:07:00 User_Input 305	Red VUHF LED	0x00110011 - 1114129		
00:00:08:00 Keyline 28	Key the mic	0x00111011 - 1118225		
00:00:09:00 Audio_In 1	Audio to be transmitted	0x00111011 - 1118225	VUHF/HF_Transmission_Out 34	34
00:00:10:00 Keyline 29	Unkey the mic	0x00110011 - 1114129		
00:00:11:00 User_Input 50	HF LED Off	0x00100000 – 1048576		
00:00:12:00 User_Input 50	VUHF LED Off	0x00000000 - 0		

Table 4. Requirement 11 Testing

The results would be identical for the AM and FM radios where the byte logic causes the upper byte to shift to be populated as the radios are selected, Similarly, the radio transmission output port are activated by the user inputs on the CSU.

As demonstrated in the results, requirement 11 is satisfied.

CSU Coupled Model DEVS Formalism

The CSU controls the selection for radio transmission and reception. It allows the operator to select or reject any audio channel both internally and externally on the aircraft through the selection command buttons. The CSU indicates the transmission and reception of radio transmission through the use of Light Emitting Diodes (LED) where yellow indicates reception, red indicates transmission and green indicates secure mode. Changes to the radio operation are accomplished through single button selection on the radio command buttons while a double click selection with 250 milliseconds commands the radio to turn off. The CSU will indicate operation whether the radios are present or not.

CSU is composed of the following atomic models:

1. CSU Interface;
2. Button Controller;
3. Crypto Controller;
4. GUI Controller; and
5. LED.

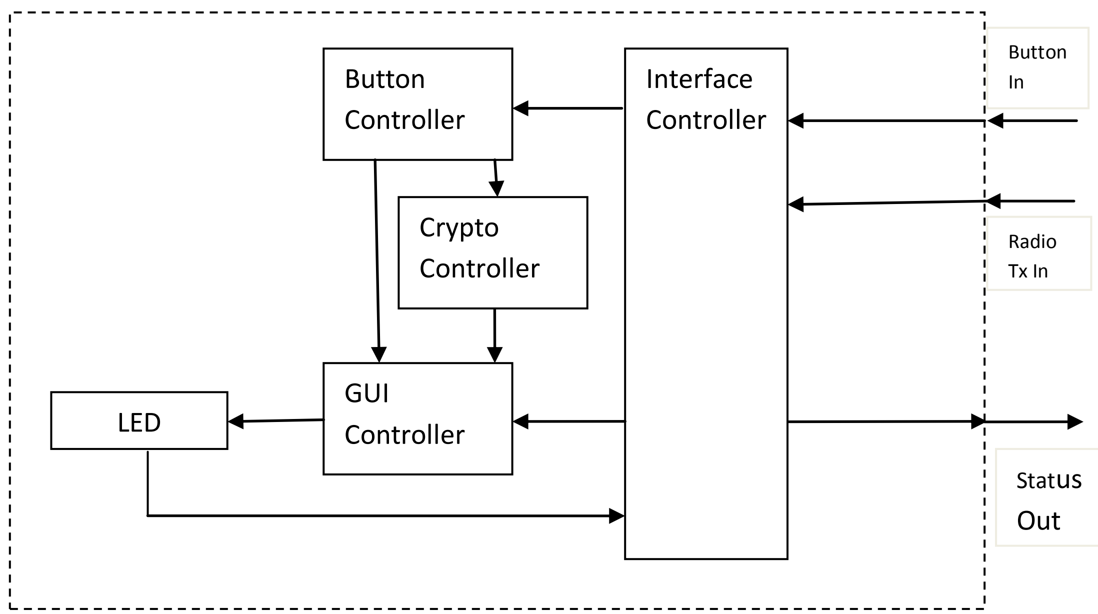

Figure 6. CSU Coupled Model

$C = < X, Y, D, \{ M_d \mid d \in D \}, EIC, EOC, IC, select >$

where X = { (Radio_Tx_In, v) | Radio_Tx_In ∈ IPorts, v ∈ N, and

(User_Input, v) | User_Input ∈ IPorts, v ∈ N };

Y = { (Status_Out, v) | Status_Out ∈ OPorts, v ∈ N };

D = { CSU_Interface, Button_Controller, Crypto_Controller, GUI_Controller, LED }

M_d = { $M_{CSU\_Interface}$, $M_{BUTTON\_Controller}$, $M_{Crypto\_controller}$, $M_{GUI\_Controller}$, M_{LED} }

EIC ⊆ ((Self, User_Input), (CSU_Interface_Controller, User_Input)),
((Self, Radio_Tx_In), (CSU_Interface_Controller, Radio_Tx_In)) }

EOC ⊆ { ((CSU_Interface_Controller, Status_Out), (Self, Status_Out)) }

IC ⊆ { ((CSU_Interface_Controller, Button_Out), (Button_Controller, Interface_In)),
((Button_Controller, Crypto_Out), (Crypto_Controller, Button_In)),
((GUI_Controller, Led_Out), (LED, Gui_In)),
((Crypto_Controller, GUI_Out), (GUI_Controller, Crypto_In)),
((Button_Controller, GUI_Out), (GUI_Controller, Button_In)),
((CSU_Interface_Controller, Gui_Out), (GUI_Controller, Interface_In)),
((LED, Led_Status_Out), (CSU_Interface_Controller, Led_Status_In)) }

select = { CSU_Interface_Controller, Button_Controller, GUI_Controller, LED, Crypto_Controller}

4.1 CSU Testing.

Testing of this coupled model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input of as demonstrated for the CMS user input:

Input	Remarks	Output Status_Out Port (Hexadecimal –Decimal)
00:00:01:00 User_input 305	Yellow LED on VUHF	0x00100001 - 1048577
00:00:02:00 User_input 305	Red LED selection on VUHF (now in transmission mode)	0x00100011 - 1048593
00:00:03:00 User_input 307	Crypto Selection on VUHF (now in secure mode)	0x00100111 - 1048849
00:00:04:00 Radio_Tx_In 1	Key the mic	0x00101111 - 1052945
00:00:05:00 User_input 306	Turn off the VUHF radio selection	0x00100100 - 1048832

Table 5. CSU Testing

The output in hexadecimal at the LED_Output port should be validated by examination as 0x00101011, which indicates the hexadecimal digit 0x00100000 as VUHF, 0x00001000 as a flashing LED state, 0x00000100 as a radio in secure mode, 0x00000010 as transmission selected, and 0x000000001 as reception capable.

As each radio is assigned a hexadecimal digit, the results should be identical with only the upper bytes changing to reflect the different radio selections. 0x00010000 is HF, 0x01000000 is FM, and 0x10000000 is AM. This is demonstrated in the log files attached to the model files.

4.2 CSU Interface Controller

The interface controller handles all input into the CSU and directs the appropriate secondary controllers to activate the logical state of the LED and respond to incoming signal such as a transmission .

$$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X = { ((Radio_Tx_In, v) | Radio_Tx_In $\in$ IPorts, v $\in$ N),

((User_Input, v) | User_Input $\in$ IPorts, v $\in$ N) }

Y = { ((Gui_Out, a) | Gui_Out $\in$ OPorts, a $\in$ N),

((Status_Out, b) | Status_Out $\in$ OPorts, b $\in$ N),

((Button_Out, c) | Button_Out $\in$ OPorts, c $\in$ N) }

S = state $\in$ { active, passive}, m_StateOfCmdButton $\in$ N, m_StateOfRadio $\in$ R, m_StateOfLed $\in$ R, m_StateOfLed $\in$ N, myState $\in$ N }

δ_{int} (User_Input | Radio_Tx_In | Led_Status_In) { passivate(); }

δ_{ext} (s, e, x)

```
{
        if(Csu_In == msg.port())
        {
                //This is the feedback loop for the CSu from a msg reception
                m_StateOfRadio = msg.value();
                myState = Keyed;

                holdIn( active, Time::Zero );

        }
        if( Led_Status_In == msg.port() )
        {
                //This is the status indication from the LED
                m_StateOfLed = msg.value();
                myState = LedInput;

                holdIn( active, Time::Zero );

        }
        if ( User_Input == msg.port() )
        {
                //Get the number representing the radio as per Declarations.h
```

```
int value = msg.value();

m_Index = (value >> 8);
int temp = (m_Index )  << 8;

//Get the action performed
int input = 0;
input = value - temp;

switch( input )
{
	case SINGLE_PRESS:
		m_StateOfCmdButton = SINGLE_PRESS;
	break;

	case DOUBLE_PRESS:
		m_StateOfCmdButton = DOUBLE_PRESS;
	break;

	case HF_CRYPTO_PRESS:
		m_StateOfCmdButton = CRYPTO_PRESS;
	break;

	case VUHF_CRYPTO_PRESS:
		m_StateOfCmdButton = CRYPTO_PRESS;
	break;
	default:
		return *this;
}

// Mask the ID into the m_StateOfCsu
if( 0 == m_Index)
	m_StateOfCmdButton |= HF;

if( 1 == m_Index)
	m_StateOfCmdButton |= VUHF;

if( 2 == m_Index)
	m_StateOfCmdButton |= FM;

if( 3 == m_Index)
	m_StateOfCmdButton |= AM;


myState = UserInput;
holdIn( active, Time::Zero );
```

```
        }

        if ( Radio_Tx_In == msg.port() )
        {

                int value = msg.value();

                if( KEYED == (short)value )
                        m_StateOfRadio = KEYED;
                else
                        m_StateOfRadio = UNKEYED;
                myState = Keyed;

                holdIn( active, Time::Zero );
        }
        return *this ;
    }
```

λ (S) {

```
if( Idle == myState)
{

}
else
{
      if( Keyed == myState )
      {
            sendOutput( msg.time(), Gui_Out, m_StateOfRadio );
            myState = Idle;
            return *this;
      }

      if( UserInput == myState )
      {
            sendOutput( msg.time(), Button_Out, m_StateOfCmdButton );
            myState = Idle;
            return *this;
      }

      if( LedInput == myState )
      {
            sendOutput( msg.time(),Status_Out, m_StateOfLed );
            myState = Idle;
            return *this;
      }
      if( LedInput == myState )
      {
            sendOutput( msg.time(),Status_Out, m_StateOfLed );
            myState = Idle;
            return *this;
      }
```

```
    }
}
```

4.2.1 CSU InterfaceTesting

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input of as demonstrated for the CMS user input

Input	Remarks	Output Gui_Out port	Output Button_Out Port	Output Status_Out Port
00:00:01:00 User_input 305	Yellow LED on VUHF		0x00100031 - 1048625	
00:00:02:00 User_input 305	Red LED selection on VUHF (now in transmission mode)		0x00100031 - 1048625	
00:00:03:00 User_input 306	Crypto Selection on VUHF (now in secure mode)		0x00100033 - 1048627	
00:00:04:00 Keyline 28	Key the mic	28		
00:00:05:00 User_input 305	Turn off the VUHF radio selection		0x00100032 - 1048626	
00:00:07:00 Keyline 29	Dekey the mic	29		
00:00:08:00 Led_Status_In 65809				0x00010111 - 65809

Table 6. CSU Interface Testing

4.3 Button Controller

The button controller interprets the user input and changes it into a command for the GUI controller to change the colour of the LED and directs the Crypto Controller to go secure .

$$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X = { (Interface_In, v) | Interface_In $\in$ IPorts, v $\in$ N where the value determines the action };

Y = { ((Gui_Out, v) | Gui_Out $\in$ OPorts, v $\in$ N),

((Crypto_Out, v) | Crypto_Out $\in$ OPorts, v $\in$ N) };

S = state $\in$ { active, passive}, m_StateOfCrypto $\in$ N, m_StateOfGui $\in$ N, m_StateOfCsu $\in$ N,

m_StateOfPermission $\in$ N, m_Index $\in$ v, input $\in$ N};

δ_{int} (Interface_In) { passivate(); }

δ_{ext} (s, e, x)

```
                {
if( Interface_In == msg.port() )
        {
                input = 0;

                //Get the number representing the radio as per Declarations.h
                //The format is upper bytes selected radio lower bytes the press -
                //double single or crypto
                int value = msg.value();

                //m_Index = (value >> 16) - 1;
                if( HF & value )
                        m_Index = 0;
                if( VUHF & value )
                        m_Index = 1;
                if( FM & value )
                        m_Index = 2;
                if( AM & value )
                        m_Index = 3;
                m_StateOfCsu[m_Index] = 0;

                int temp = value & 0xffff0000;

                //Get the action performed
                input = value - temp;
                switch(input)
                {
                                case SINGLE_PRESS:
                                        //Off to yellow to red continuously
```

```
					m_StateOfGui[m_Index] = (short)(m_StateOfGui[m_Index]  + 1 )
% 3;
					if( 0 == m_StateOfGui[m_Index] )
						m_StateOfGui[m_Index] = 1;
					if( 2 == m_StateOfGui[m_Index] )
						m_StateOfPermission[m_Index]  = 1;

				break;

				case DOUBLE_PRESS:
					m_StateOfGui[m_Index]  = 0;
					m_StateOfPermission[m_Index]  = 0;
				break;

				case CRYPTO_PRESS:
					// Crypto can be on or off depending on the user input
					m_StateOfCrypto[m_Index] =
(short)(m_StateOfCrypto[m_Index]  + 1) % 2;
					m_StateOfPermission[m_Index] = m_StateOfCrypto[m_Index];
					if( 0 == m_StateOfCrypto[m_Index] )
						m_StateOfPermission[m_Index]  = 0;

				break;
		}

		if( 0 == m_Index)
		{
				m_StateOfCsu[m_Index] |= HF;
				int temp = (short)m_StateOfGui[m_Index] +
(short)m_StateOfCrypto[m_Index];

				m_StateOfCsu[m_Index] |= temp;
				m_StateOfCrypto[m_Index] |= HF;
		}
		else if( 1 == m_Index)
		{
				m_StateOfCsu[m_Index] |= VUHF;
				int temp = (short)m_StateOfGui[m_Index] +
(short)m_StateOfCrypto[m_Index];
				m_StateOfCsu[m_Index] |= temp;
				m_StateOfCrypto[m_Index] |= VUHF;
		}
		else if( 2 == m_Index)
		{
				m_StateOfCsu[m_Index] |= FM;
				m_StateOfCsu[m_Index] |= m_StateOfGui[m_Index];
		}
```

```
                else if( 3 == m_Index)
                {
                                m_StateOfCsu[m_Index] |= AM;
                                m_StateOfCsu[m_Index] |= m_StateOfGui[m_Index];
                }


                holdIn( active, Time::Zero );

        }

}
        λ (S) {
                if( 65535 < m_StateOfCsu[m_Index] && 3 != ( short )( m_StateOfCsu[m_Index] ) )
                {
                // The format is 0x00010002 meaning HF is transmitting
                sendOutput(msg.time(), Gui_Out, m_StateOfCsu[m_Index]);
                }

                //Let crypto know that there was a button press to send the crypto state
                //which is independent of the user command button selection of a single
                //or double press
                if( CRYPTO_PRESS == input && 1 == m_StateOfPermission[m_Index])
                {
                sendOutput(msg.time(), Crypto_Out, m_StateOfCrypto[m_Index]);
                }
            }
```

4.3.1 Button Controller Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as follows:

Input	Remarks	Output Gui_Out port	Output Crypto_out port
00:00:01:00 Interface_In 304	Yellow LED on VUHF	0x00010001	
00:00:02:00 Interface_In 304	Red LED selection on VUHF (now in transmission mode)	0x00010002	
00:00:03:00 Interface_In 306	Crypto Selection on VUHF (now in secure mode)	0x00010003	0x00010001
00:00:04:00 Interface_In 306	Crypto Selection on VUHF (now in unsecure mode)	0x00010003	0x00010000

00:00:05:00 Interface_In 305	Turn off the VUHF radio selection	0x00010000	

Table 7. Button Controller Testing

4.4 Crypto Controller

The crypto controller retains the crypto state and distributes the permission to go secure or not.

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$

$X = \{ (Button\_In, v) \mid Button\_In \in IPorts, v \in R \};$

$Y = \{ ((Gui\_Out, a) \mid Gui\_Out \in OPorts, a \in R \};$

$S = state \in \{ active, passive\}, m\_StateOfCrypto \in N, m\_Index \in N \}$

```
δint (Button_In) { passivate(); }
δext (s, e, x) {
        if( Button_In == msg.port() )
         {
                        int value = msg.value();

                        if( value & HF )
                                m_Index = 0;
                        if( value & VUHF )
                                m_Index = 1;

                        //Crypto can be on or off depending on the user input
                        m_StateOfCrypto[m_Index]  = value;

                        myState = Rx;
                        holdIn( active, Time::Zero  ) ;
                }
         }
λ (S){
        if( Rx == myState )
        {
        sendOutput( msg.time(),Gui_Out, m_StateOfCrypto[m_Index]);
        myState = Idle;
        }
}
```

4.4.1 Crypto Controller Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as follows:

Input	Remarks	Output GUi_Out port
00:00:01:00 Button_In 0x10001	HF in secure mode	0x10001
00:00:01:00 Button_In 0x10001	HF in unsecure mode	0x10001

Table 8. Crypto Controller Testing

4.5 GUI Controller

The GUI controller interfaces with the LED and commands it to change colour based on its inputs. A byte packed input with non-zero lower bytes from the crypto controller means secure, a byte packed input with non-zero input from the CSU interface could mean a radio transmission is occurring depending on the byte pattern, and the input from the button controller identifies the state of the LED as it would be presented to the user.

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$

X = { ((Button_In, v) | Button_In $\in$ IPorts, v $\in$ N),

((Crypto_In, v) | Crypto_In $\in$ IPorts, v $\in$ N),

((Interface _In ,v) | Interface_In $\in$ IPorts, v $\in$ N) };

Y = { ((LED_Out, v) | Gui_Out $\in$ OPorts, v $\in$ N };

S = state $\in$ { active, passive}, m_StateOfLed $\in$ N, m_StateOfCrypto $\in$ N,

m_Index $\in$ N, nAction $\in$ N, myState $\in$ N }

δ_{int} (Button_In | Crypto_In | Interface_In) { passivate(); }

δ_{ext} (s, e, X)

```
        {

        //Doesn't matter what the other interfaces send until the button controller sends
information
    if(Button_In == msg.port())
    {
        int value = msg.value();
        nAction = short(msg.value());
                if( 3 == nAction )
            return *this;

        int temp = 0;

        if(value & HF)
            m_Index = 0;
        if( value & VUHF)
            m_Index = 1;
```

```
        if( value & FM)
                m_Index = 2;
        if( value & AM)
                m_Index = 3;

        m_StateOfLed[m_Index] = msg.value();


        //Button Controller told us which radio in the upper byte
        //and what the Led lighting effect is to be
        //0 means no Led  on
        //1 means yellow
        //2 means red
        //3 means crypto

        //Get the index

        if( 0 != nAction )
        {
                // yellow to red to yellow to red etc.
                temp = (short)( m_StateOfLed[m_Index] );
        }
        else
                m_StateOfLed[m_Index] &= 0xffff0000;

        //Pack the information for sending
         m_StateOfLed[m_Index] &= 0xffff0000;

         // | AM | FM |VUHF| HF |  F | Cr | Tx | Rx |
        if( 1 == temp )
        {
                m_StateOfLed[m_Index] |= 0x1;
        }
        if( 2 == temp )
        {
                m_StateOfLed[m_Index] |= 0x11;
        }
        myState = Button;
        holdIn( active, Time::Zero  ) ;

}

if( Crypto_In == msg.port() )
{
        int value = msg.value();

        if ( 0 < m_StateOfLed[m_Index] )
```

```
			m_StateOfCrypto[m_Index] = value;

		if( 1 == (short)m_StateOfCrypto[m_Index] )
		{
			m_StateOfLed[m_Index] |= 0x100;
		}
		else
		{
			m_StateOfLed[m_Index] &= ~0x100;
		}

		myState = Crypto;

		holdIn( active, Time::Zero  ) ;

	}

	if( Interface_In == msg.port() )
	{
		int value = msg.value();
		int nRadioSelected = 0;
		if( HF & value )
			m_Index = 0;
		if( VUHF & value )
			m_Index = 1;
		if( FM & value )
			m_Index = 2;
		if( AM & value )
			m_Index = 3;

		if ( UNKEYED == (short)value )
		{
			//Return to previous unflashing state
			m_StateOfLed[m_Index] &= ~0x0000f000;
		}
		else
		{
			//Report a flashing state if LED is active
			if(0 < (short)m_StateOfLed[m_Index] )
			{
				m_StateOfLed[m_Index] |= 0x1000;
			}
		}

		myState = Interface;

		holdIn( active, Time::Zero  ) ;
```

```
} }
λ (S) {    if( Idle != myState)
           {
                      sendOutput( msg.time(), Led_Out, m_StateOfLed[m_Index]);
                      myState = Idle;
           }
 }
```

4.5.1 GUI Controller Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as detailed in the following table, an output would be generated as follows:

Input	Remarks	Output Led_Out port (Hexadecimal – Decimal)
00:00:00:700 interface_In 0x00010028	Keyline active	0x00010000 - 65536
00:00:00:800 Button_In 0x00010031	Single press on the HF – Reception	0x00010001 - 65537
00:00:00:90 Interface_In 0x00010000	Keyline inactive	0x00010000 - 65536
00:00:01:00 Interface_In 0x00010028	Keyline active	0x00010000 - 65536
00:00:02:00 Button_In 0x00010031	Single press on the HF - Transmission	0x00011011 - 69649
00:00:03:00 interface_In 0x00010000	Keyline inactive	0x00010011 - 69649
00:00:04:00 Crypto_In 0x00010001	Crypto selection active	0x00010111 - 65809
00:00:05:00 interface_In 0x00010028	Keyline active	0x00011111 - 69905
00:00:06:00 interface_In 0x00010000	Keyline inactive	0x00010111 - 65809
00:00:07:00 Button_In 0x00010000	Double press on the HF	0x00010000 - 65536
00:00:08:00 interface_In 0x00010028	Keyline active	0x00010000 - 65536
00:00:09:00 interface_In 0x00010000	Keyline inactive	0x00010000 - 65536

Table 9. GUI Controller Testing

The output would similar with only the upper bytes changing for the respective radios as detailed in Declarations.h.

4.6 LED

The Led (actually in real life there are 12 X 39 X 3 = 1404 LEDs total) that toggle from off to either red, yellow of green depending on the command input.

$$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

```
X = { (Gui_In, v ) | Gui_In ∈ IPorts, v ∈ N};
Y = { (LED_Status_Out, v ) | LED_Status_Out ∈ OPorts, v ∈ N };
S =  state ∈ { active, passive},  m_StateOfLed ∈ N, m_OldStateOfLed ∈ N }
δint (Gui_In) { passivate(); }
δext (s,e,x ) {
            if(Gui_In == msg.port() )
            {
                //The GUI controller sent a message with the appropriate
                //radio head, the crypto, and the states to be
                //A bit manipulation work then be done but the Led is just going
                //to affirm that the action was done and send the GUI Controller's
                //message to the output.
                m_StateOfLed = ( msg.value());

                holdIn( active, Time::Zero  ) ;
            }
    }
λ (S)   {
    //In reality the bitmap to display Led colour depending on state would be seen
    //but because Devs is a black box, send the output to the interface for output
    //where a fully packed data value is successful
    //Check to see if it is sent to prevent run away as the LED is always on

    if( m_OldStateOfLed != m_StateOfLed )
    {
        sendOutput (msg.time(), Led_Status_Out, m_StateOfLed);
        m_OldStateOfLed = m_StateOfLed;
    }
    }
```

4.6.1 LED Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as detailed in the following table, an output would be generated as follows:

Input	Remarks	Output Led_Out port (Hexadecimal – Decimal)
00:00:01:00 Gui_In 0x00010000	HF selected but not active	0x00010000 - 65536
00:00:02:00 Gui_In 0x00010001	Single press on the HF – Reception	0x00010001 - 65537
00:00:03:00 Gui_In 0x00010011	Single press on the HF – Transmission	0x00010011 - 65553
00:00:04:00 Gui_In 0x00010001	Single press on the HF – Reception	0x00010001 - 65537
00:00:05:00 Gui_In 0x00010011	Single press on the HF - Transmission	0x00010011 - 65553
00:00:06:00 Gui_In 0x00010111	Secure mode	0x00010111 - 65809
00:00:07:00 Gui_In 0x00011111	Transmission occuring	0x00011111 - 69905
00:00:08:00 Gui_In 0x00010101	Single press on the HF – Secure Reception	0x00010101 - 65793
00:00:09:00 Gui_In 0x00010000	Double press	0x00010000 - 65536

Table 10. LED Testing

5 HF Coupled Model DEVS Formalism

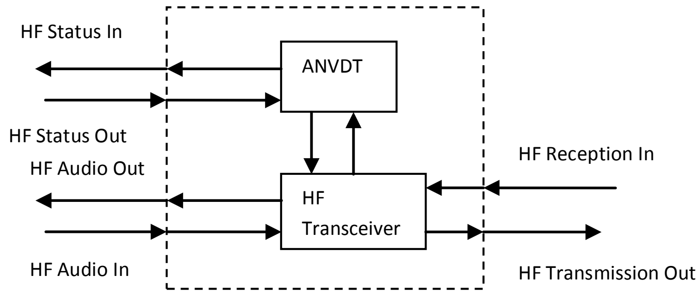

Figure 7. HF Coupled Model

C = < X, Y, D, { M_d | d ϵ D }, EIC, EOC, IC, select >

where X = { ((HF_Status_In, v) | HF_Status_In ∈ IPorts, v ∈ N),
((HF_Audio_In, v) | HF_Audio_In ∈ IPorts, v ∈ N),
((HF_Reception_In, v) | HF_Reception_In ∈ IPorts, v ∈ N) };

Y = { (ANVDT_Status_Out, v) | ANVDT_Status_Out ∈ IPorts, v ∈ N),
(HF_Audio_Out, v) | HF_Audio_Out ∈ IPorts, v ∈ N),
((HF_Transmission_Out, v) | HF_Transmission_Out ∈ IPorts, v ∈ N) };

D = { ANVDT, HF_Transceiver }

M_d = { M_{ANVDT}, $M_{HF\_Transceiver}$ }

EIC ⊆ { ((CCU, HF_Status_In), (ANVDT, HF_Status_In)),
((CCU, HF_Audio_In), (HF_Transceiver, Audio_In))
((Self, HF_Reception_In), (HF_Transceiver, HF_Reception_In)) }

EOC ⊆ { ((ANVDT, HF_Status_Out), (Self, ANVDT_Status_In)),
((HF_Transceiver, HF_Audio_Out), (Self, HF_Audio_Out)),
((HF_Transceiver, HF_Transmission_Out), (Self, HF_Transmssion_Out)) }

IC ⊆ { ((ANVDT, HF_Tranceiver_Out), (HF_Transceiver, ANVDT_In)),
((HF_Transceiver, ANVDT_Out), (ANVDT, HF_Transceiver_In)); }

select = { HF_Transceiver, ANVDT }

5.1 HF Testing

Testing of this coupled model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input of as demonstrated as follows:

Input	Remarks	Output Status_Out Port (Hexadecimal –Decimal)		
00:00:01:00 HF_Status_In 0	Requesting Hf status	0x21 if online(76.1% mean)		
00:00:02:00 HF_Transceiver_In 38	Msg for encoding / decoding	If online, 0x2A		
		If offline, 0x23		
		Output HF_Audio_Out port	Output Hf_Transmission_Out Port	Output ANVDT_Out Port
00:00:03:00 ANVDT_In 34	Decoded message data	0x22		
00:00:04:00 ANVDT_In 42	Encoded message		0x22	
00:00:05:00 Audio_In 34	Audio to be transmitted			0x26
00:00:05:00 HF_Reception_In 40	Valid Hf Reception Message			0x27

Table 11. HF Testing

5.2 ANVDT

The Advanced Narrowband Secure Voice Terminal (ANDVT) has been the workhorse secure voice terminal for low bandwidth secure voice communications throughout NATO and is used for secure HF radio transmission. The ANVDT is a separate unit from the HF transceiver and handles the encryption and decryption of the HF signal.

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$

X = { ((HF_Status_In, u) | HF_Status_In $\in$ IPorts, u $\in$ R),

((HF_Transceiver_In, v) | HF_Transceiver_In $\in$ IPorts, v $\in$ N) };

Y = { ((HF_Status_Out, u) | HF_Status_Out $\in$ OPorts, u $\in$ R),

((HF_Transceiver_Out, v) | HF_Transceiver_Out $\in$ OPorts, v $\in$ N) };

S = state $\in$ { active, passive}, m_StateOfHf $\in$ N }

δ_{int} (HF_Status_In | HF_Transceiver_In) { passivate(); }

δ_{ext} (s, e, x)

```
{
if( HF_Status_In == msg.port() )
{

if( 0 ==  msg.value() )
{
```

```
			//Figure out if the HF is good so check by setting it as a random
			//number distribution where 0 to 0.75 is good
			fRandNumber = rand()/(float)RAND_MAX;
			if( 0.25 < fRandNumber )
				m_StateOfHf = OPERATIONAL;
			else
				m_StateOfHf = NONOPERATIONAL;

			//Testing remove comment
			//m_StateOfHf = OPERATIONAL;


			}
			holdIn( active, Time::Zero );
		}

		if( HF_Transceiver_In == msg.port() )
		{
		if( DECODED ==  msg.value() )
		{
			//HF transmission so encrypt/decrypt if online
			if( NONOPERATIONAL == m_StateOfHf )
				m_StateOfHf = NO_AUDIO;
			else if( OPERATIONAL == m_StateOfHf)
				m_StateOfHf = DECODED;


		}
		else if( ENCODED ==  msg.value() )
		{

			//HF transmission so encrypt/decrypt if online
			if( NONOPERATIONAL == m_StateOfHf )
				m_StateOfHf = NO_AUDIO;
			else if( OPERATIONAL == m_StateOfHf)
				m_StateOfHf = ENCODED;

		}

		holdIn( active, Time::Zero );
		} `
}

λ (S)	{
			if( DECODED  == m_StateOfHf  || ENCODED  == m_StateOfHf )
			{
				sendOutput( msg.time(), HF_Transceiver_Out, m_StateOfHf);
```

```
        }

        if( OPERATIONAL == m_StateOfHf )
        {
                sendOutput( msg.time(), HF_Status_Out, m_StateOfHf);
        }
}
```

5.3.1 ANVDT Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as detailed in the following table, an output would be generated as follows:

Input	Remarks	Output HF_Status_Out port
00:00:01:00 HF_Status_In 0	Requesting Hf status	0x21 if online(76.1% mean)
00:00:02:00 HF_Transceiver_In 0	Msg for encoding / decoding	If online, 26
		If offline, 23

Table 12. ANVST Testing

5.3 HF_Transceiver

The airborne HF-121C radio set, nomenclature AN/ARC-512(V), is the next generation of the HF-121/121A/121B radios designed for military voice, data and Link 11 applications. Only voice applications will be modelled. The HF Transceiver receives outside transmission and sends it to the ANVDT for decoding. It also receives the transmission from the ANVDT that is either secure or unsecure depending whether it is to be transmitted or placed on the audio channel .

M = < X, Y, S, δ_{int}, δ_{ext}, λ, ta >

X = { ((Audio_In, v) | Audio_In $\in$ IPorts, u $\in$ N),

((ANVDT_In,vu) | ANVDT_In $\in$ IPorts, u $\in$ N),

((HF_Reception_In, v) | HF_Reception_In $\in$ IPorts, u $\in$ N) };

Y = { (HF_Audio_Out, v) | HF_Audio_Out $\in$ OPorts, v $\in$ R),

((ANVDT_Out, v) | ANVDT_Out $\in$ OPorts, v $\in$ R),

((HF_Reception_Out, v) | HF_Reception_Out $\in$ OPorts, v $\in$ R)};

S = state $\in$ { active, passive}, m_StateOfHf $\in$ R, m_RadioTx $\in$ N, myState $\in$ N }

δ_{int} (Audio_In | ANVDT_In | HF_Reception_In) { passivate(); }

```
δext (s, e, x)
            {
m_RadioTx = 0;

            if( msg.port() == ANVDT_In )
            {

                        m_StateOfHf = msg.value();


                        if(DECODED  == m_StateOfHf || ENCODED  == m_StateOfHf )
                        {
                                    m_RadioTx = AUDIO;
                                    myState = Relay;
                        }
                        else
                                    m_RadioTx = NO_AUDIO;
                        myState = Relay;

                        holdIn( active, Time::Zero );

            }

            if( Audio_In == msg.port() )
            {
                        //We have valid audio and a keyline so generate
                        //an output and send it to the ANVDT for encryption

                        m_RadioTx = TRANSMISSION;
                        myState = Tx;
                        holdIn( active, Time::Zero );


            }
            if( HF_Reception_In == msg.port() )
            {
                        //We have a message in so send so store it and send it to the
                        // ANVDT for decryption
                        m_RadioTx = DECODED;
                        myState = Rx;

                        holdIn( active, Time::Zero );
            }
}


λ (S)     {
```

```
		//Relay over to the CCU for audio through the headsets on the Audio_Out port

		if( Rx == myState )
		{
			sendOutput(msg.time(), ANVDT_Out, m_RadioTx);
		}
		if( Tx == myState )
		{
			sendOutput(msg.time(), ANVDT_Out, m_RadioTx);
		}

		if( Relay == myState )
		{
			if( AUDIO == m_RadioTx && DECODED == m_StateOfHf)
			{
				sendOutput( msg.time(), HF_Audio_Out, m_RadioTx);
			}
			else if( AUDIO == m_RadioTx && ENCODED == m_StateOfHf)
			{
				sendOutput( msg.time(), HF_Transmission_Out, m_RadioTx);
			}
		}
	}
```

5.3.1 HF_Transceiver Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and then examining the output.

Thus with an input as follows, and output can be generated. Note, however, that the Audio_In port is not checked for range or zero value because the CCU has valid the requirements and therefore any input on this port will be transmitted and produces a value of 34 as defined in the Declarations.h file. Any values other than the values specified on the ANDVT_In will not produce an output, while any input on HF_Reception_In will generated a value of 39 on the ANVDT_Out port.

Input	Remarks	Output HF_Audio_Out port	Output Hf_Transmission_Out Port	Output ANVDT_Out Port
00:00:01:00 ANVDT_In 27	Decoded message data	0x22		
00:00:02:00 ANVDT_In 30	Encoded message		0x22	
00:00:03:00 Audio_In 34	Audio to be transmitted			0x26
00:00:04:00 HF_Reception_In 40	Valid Hf Reception Message			0x27

Table 13. HF Transceiver Testing

6 CCU Coupled Model DEVS Formalism

The CCU is the heart of the CMS and controls and passes audio and status messages bidirectionally through the CMS. The CCU communicates to the individual VUHF radios through a Mil-Std-1553B interface while the HF radios are communicated through discrete interfaces. The AM and FM radios are non-secure and are also operated through a discrete interface with the CCU. Keyline and audio channels are directly routed through the CCU to the applicable radio and CSU. Of note, the CCU is controlled and responds to Mil-Std-1553B messages from one of the four Crew Display Units (CDU) which is not being modelled.

CCU is composed of the following atomic models:

1. CCU Interface;
2. Status Controller; and
3. Audio Multiplexer.

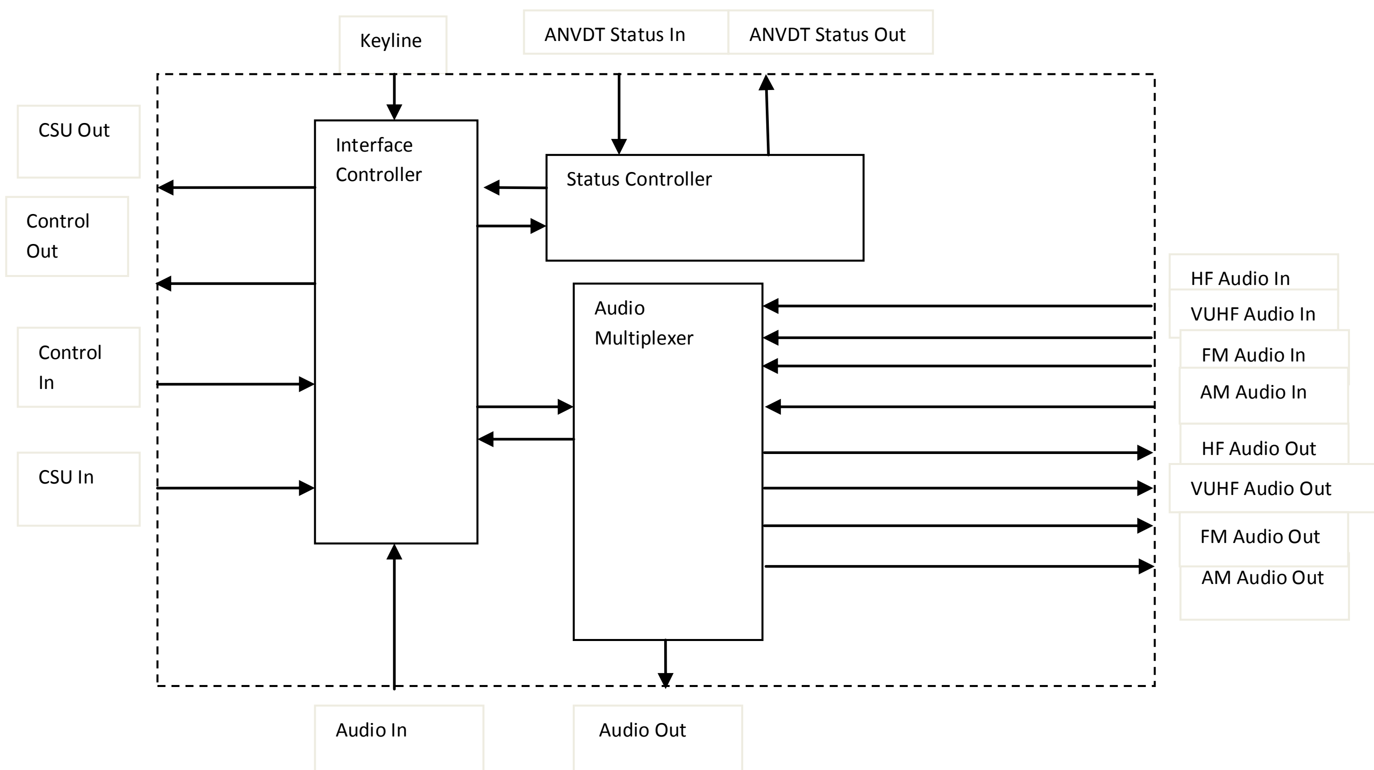

Figure 8. CCU Coupled Model

C = < X, Y, D, { M_d | d ϵ D }, EIC, EOC, IC, select >

where X = { ((HF_Audio_In, v) | HF_Audio_In ∈ IPorts),
((ANDVT_Status_In, v) | ANVDT_Status_In ∈ IPorts),
((VUHF_Audio_In, v) | VUHF_Audio_In ∈ IPorts),
((FM_Audio_In, v) | FM_Audio_In ∈ IPorts),
((AM_Audio_In, v) | AM_Audio_In ∈ IPorts),
((Keyline, v) | Keyline ∈ IPorts),
((CSU_In, v) | CSU_In ∈ IPorts),
((Audio_In, v) | Audio_In ∈ IPorts) }

Y = { ((HF_Audio_Out, v) | HF_Audio_Out ∈ OPorts),
((ANVDT_Status_Out, v) | ANVDT_Status_Out ∈ OPorts),
((VUHF_Audio_Out, v) | VUHF_Audio_Out ∈ OPorts),
((FM_Audio_Out, v) | FM_Audio_Out ∈ OPorts),
((AM_Audio_Out, v) | AM_Audio_Out ∈ OPorts),
((CSU_Out, v) | CSU_Out ∈ OPorts),
((Audio_Out, v) | Audio_Out ∈ OPorts) }}

D = { CCU_Interface, Status_Controller, Audio_Multiplexer }

M_d = { $M_{CCU\_Interface}$, $M_{Status\_Controller}$, $M_{Audio\_Controller}$ }

EIC ⊆ { ((Self, Audio_In), (CCU_Interface, Audio_In)),
((Self, Keyline), (CCU_Interface, Keyline_In)),
((Self, CSU_In), (CCU_Interface, CSU_In)),
((Self, ANVDT_Status_In), (Status_Controller, ANVDT_In)),
((Self, HF_Audio_In), (Audio_Multiplexer, HF_Audio_In)),
((Self, VUHF_Audio_In), (Audio_Multiplexer, VUHF_Audio_In)),
((Self, FM_Audio_In), (Audio_Multiplexer, FM_Audio_In)),
((Self, AM_Audio_In), (Audio_Multiplexer, AM_Audio_In)) }

EOC ⊆ { ((Audio_Multiplexer, FM_Audio_Out), (Self, FM_Audio_Out)),
((Audio_Multiplexer, VUHF_Audio_Out), (Self, VUHF_Audio_Out)),
((Audio_Multiplexer, HF_Audio_Out), (Self, HF_Audio_Out)),
((Audio_Multiplexer, AM_Audio_Out), (Self, AM_Audio_Out)),
((CCU_Interface, Reception_Out), (Self, Audio_Out)),
((CCU_Interface, CSU_Out), (Self, CSU_Out)),
((Status_Controller, ANVDT_Out), (Self, ANVDT_Status_Out)) }

IC ⊆ { ((CCU_Interface, Status_Out), (Status_Controller, Interface_In)),
((CCU_Interface, Audio_Out), (Audio_Multiplexer, Audio_In));
((Status_Controller, Interface_Out), (CCU_Interface, Status_In);
((Audio_Multiplexer, Audio_Out), (CCU_Interface, Reception_In)) }

select = { CCU_Interface, Status_Controller, Audio_Multiplexer}

6.1 CCU Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input of as demonstrated as follows:

Input	Remarks	Output Audio_Out port	Output Status_Out Port	Output CSU_Out Port
00:00:01:00 Status_In 0	Unknown status of the radios on initial operation		0	
00:00:02:00 Status_In 286326784	All radios are polled for a maximum of three times or until an operational result is received			
00:00:03:00 Keyline 40	Active microphone, pass through radio active and keyline	0x11110001		0x1111028 – 286326824
00:00:04:00 Keyline 41	Deactivate the microphone	0x11110000		0x11110029 - 286326825
00:00:05:00 CSU_In 0	No valid user input, it will send only send out msgs if the correct parameters are set such as audio, keyline or active transmitting CSU selection			
00:00:06:00 Keyline 40	Reactivate the microphone- need a valid key as one of the conditions for output.	0x11110001		0x111100028 - 286326824
00:00:07:00 CSU_In 65553	Allow the CCU to see that the CSU is able to transmit on HF	0x11110011		
00:00:08:00 Audio_In 34	Valid audio Signal	0x11110111		
00:00:09:00 Audio_In 35	Invalid audio Signal	0x11110101		
00:00:10:00 Audio_In 34	Valid audio Signal	0x11110111		
00:00:10:00 Keyline 41	Unkeyed Signal	0x11110110		0x11110029

Input	Remarks	Output – Audio _Speaker_Out	Output- CSU_Out
00:00:01:00 Am_Reception_In 40	AM Transceiver reception message	0x10000028 – 268439552 Audio_Out	
00:00:02:00 Fm_Reception_In 40	FM Transceiver reception message	0x01000028 – 16777472 Audio_Out	
00:00:03:00 Vuhf_Reception_In 40	VUHF Transceiver reception message	0x00100028 – 1048592 Audio_Out	
00:00:04:00 HF_Reception_In 40	HF reception message	0x00010028 – 65537 Audio_Out	

Input	Remarks	Output Anvdt_Out port	Output Interface_Out Port (Hexadecimal – Decimal)
Returns the status of the HF radio until ANVDT reports operational		0	
00:00:01:00 ANVDT_In 33	Returns the state of the radios.	0x1111000 0 - 286326784	
00:00:02:00 ANVDT_In 21	Bad data and nothing will happen		

Input	Remarks	Output Port			
00:00:05:00 Audio_In 0	No active transmitters, no key, no audio, transmission selected				
00:00:06:00 Audio_In 65809	Keyline, transmitter selected, audio for HF	34 HF_Audio_Out			
00:00:07:00 Audio_In 1048849	Keyline, transmitter selected, audio for VUHF		34 VUHF_Audio _Out		
00:00:08:00 Audio_In 16777489	Keyline, transmitter selected, audio for FM			34 FM_Audio_out	
00:00:09:00 Audio_In	Keyline, transmitter				34 AM_Audio_Ou

268435729	selected, audio for AM				t
00:00:10:00 Audio_In 286327057	Keyline, transmitter selected, audio for all radios	34 HF_Audio_Out	34 VUHF_Audio _Out	34 FM_Audio_Ou t	34 AM_Audio_Ou t

Table 14. CCU Testing

6.2 CCU Interface

The CCU interface handles all input into the CCU and directs the input and output to the appropriate controller for radio communication.

$$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X = { ((Audio_In, v) | Audio_In ∈ IPorts, v ∈ R),

((Keyline_In, v) | Keyline_In ∈ IPorts, v ∈ R),

((CSU_In, v) | CSU_In ∈ IPorts, v ∈ R),

((Status_In, v) | Status_In ∈ IPorts, v ∈ R) }

Y = { ((Status_Out, b) | Status_Out ∈ OPorts, b ∈ R)

((Audio_Out, b) | Audio_Out ∈ OPorts, b ∈ R),

((CSU_Out, b) | CSU_Out ∈ OPorts, b ∈ R) }

S = state ∈ { active, passive}, m_StateOfCsu ∈ N, m_StateOfControl ∈ N, m_StateOfStatusControl ∈ N, m_StateOfKeyline ∈ N, m_StateOfAudioIn ∈ N, m_StateOfAudio ∈ N }

δ_{int} = { (Audio_In | Keyline_In | CSU_In | Status_In) passivate(); }

δ_{ext} (s,e,x) = {

```
m_StateOfControl = 0;
// m_StateOfControl is a dense populated variable where a 1 per byte
//indicates operational. Thus 0x11110111 means all radios, not used,
//audio, valid CSU
//transmission selection and a keyline respectively.
if( Keyline_In == msg.port() )
{
        m_StateOfKeyline = msg.value();
        if( 0 != m_StateOfKeyline )
                m_StateOfControl |= 0x1;
```

```
		else m_StateOfControl &= ~0x1;
	}
	if( CSU_In  == msg.port() )
	{
		//The data is the radios in the upper bytes and the LEDs status
		//in the lower bytes. Right now, the interface is concerned with
		// the lower selection. I.e. whether the LED is off.
		//
		if( 0 != (short)msg.value() )
			m_StateOfCsu = msg.value();
		else
			m_StateOfCsu = 0;

		//Transmitting
		if( 0x10 == (short)msg.value() && 0x10 )
			m_StateOfControl  |= 0x10;
		else
			m_StateOfControl &= ~0x10;

	}

	if( Status_In  == msg.port() )
	{
		//The Status Controller packs the upper bytes with the
		//status of the radios, so retrieve the states.
		m_StateOfStatusControl = msg.value();

		//Populate the radios states compared to the CSU
		//Remember the CSU can select a radio even if it is not working
		//first shift the two variables down to the lower bytes to ensure
		//no extra information is added

		int temp1 = 0;
		int temp2 = 0;
		int temp3 = 0;

		temp1 = m_StateOfCsu >> 16;
		temp2 = m_StateOfStatusControl  >> 16;
		temp3 = temp1 & temp2;

		temp3 = temp3 << 16;

		m_StateOfControl |= temp3;
	}

	if( Audio_In == msg.port() )
	{
```

```
                    m_StateOfAudio = msg.value();
                    if( 0 != m_StateOfAudio)
                            m_StateOfControl |= 0x100;
                    else m_StateOfControl &= ~0x100;

                }
            }

λ (S) = {
        //Check for valid radios
        if(!m_StateOfStatusControl)
        {
            //Get them
            sendOutput( msg.time(), Status_Out, m_StateOfControl);
        }

        if( 0 != (short)m_StateOfControl )
        {
            sendOutput( msg.time(), CSU_Out, m_StateOfControl );
        }

        if( m_StateOfAudio && m_StateOfKeyline && m_StateOfCsu )
            sendOutput( msg.time(), Audio_Out, m_StateOfControl);
  }
```

6.2.1 CCU Interface Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input of as demonstrated for the CMS user input

Input	Remarks	Output Audio_Out port	Output Status_Out Port	Output CSU_Out Port
00:00:01:00 Status_In 0	Unknown status of the radios on initial operation		0	
00:00:02:00 Status_In 286326784	All radios on operational			
00:00:03:00 Keyline 1	Active the microphone			
00:00:04:00 Keyline 0	Deactivate the microphone			
00:00:05:00 CSU_In 0	No valid user input, it will send out the status variable as configured so far if there is no audio, keyline or active transmitting CSU selection			0x11110000 - 286326784
00:00:06:00 Keyline 1	Reactivate the microphone-			

	need a valid key as one of the conditions for output			
00:00:07:00 CSU_In 65553	Allow the CCU to see that the CSU is able to transmit on HF			
00:00:08:00 Audio_In 1	Valid audio Signal	0x11110111		
00:00:09:00 Audio_In 1	Invalid audio Signal			
00:00:10:00 Audio_In 1	Valid audio Signal	0x11110111		

Table 15. CCU Interface Testing

6.3 Status Controller

The status controller handles the status of the radios and outputs a request to the radios to respond indicating that the radios are present and operational.

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$

X = { ((AM_In, v) | AM_In ∈ IPorts, v ∈ R),
((ANVDT_In, v) | ANVDT_In ∈ IPorts, v ∈ N),
((FM_In, v) | FM_In ∈ IPorts, v ∈ N),
((VUHF_In, v) | VUHF_In ∈ IPorts, v ∈ N),
((Interface_In, v) | Interface_In ∈ IPorts, v ∈ N) }

Y = { ((AM_Out, v) | AM_Out ∈ OPorts, v ∈ R),
((ANVDT_Out, v) | ANVDT_Out ∈ OPorts, v ∈ N),
((FM_Out, v) | FM_Out ∈ OPorts, v ∈ N),
((VUHF_Out, v) | VUHF_Out ∈ OPorts, v ∈ N),
((Interface_Out, v) | Interface_Out ∈ OPorts, v ∈ N) }

S = state ∈ { active, passive}, m_AnvdtState ∈ N, m_RadioState ∈ N, }

δ_{int} = { (AM_In | ANVDT_In |FM_In | VUHF_In | Interface_In) passivate(); }
δ_{ext} (s,e,x) = {

```
                Sigma = generate random Processing Time;
if(Interface_In == msg.port() )
    {
        if(Interface_In == msg.port() )
        {
```

```
            //The CCU Interface sent a 0 to inform us that there is no status on the //radios
            if( 0 == msg.value() )
            {
                float fRandNum = 0;
                int temp[3] = {0};
                for (int i =0; i <3 ;i++)
                {
                    fRandNum  = rand()/(float)RAND_MAX;
                    if(0.75 < fRandNum )
                        temp[i] = NONOPERATIONAL;
                    else
                        temp[i] = OPERATIONAL;

                }
                //AM
                if(OPERATIONAL == temp[0])
                        m_RadioState |= AM;
                //FM
                if(OPERATIONAL == temp[1])
                        m_RadioState |= FM;
                //VUHF
                if(OPERATIONAL == temp[2])
                        m_RadioState |= VUHF;
            }
            holdIn( active, Time::Zero );

            }

            if( ANVDT_In == msg.port() )
            {
                m_AnvdtState = msg.value();
                if( OPERATIONAL == m_AnvdtState)
                    m_RadioState |= HF;
                else
                    m_RadioState &= ~HF;

                holdIn( active, Time::Zero );

            }
        }
λ (S) = {
            if( OPERATIONAL != m_AnvdtState)
                sendOutput( msg.time(), ANVDT_Out, (short)m_AnvdtState );
            else
            {
            sendOutput( msg.time(), Interface_Out, m_RadioState );
            }
```

}

6.4.1 Status Controller Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus testing can be accomplished with an input as follows. Note that the state of the AM Transceiver, FM Transceiver and VUHF Transceiver are randomly set as operational. Hence, an output on the Interface_Out port requires HF to be operational but the same cannot be said of the other transceivers.

Input	Remarks	Output Anvdt_Out port	Output Interface_Out Port (Hexadecimal – Decimal)
	Returns the status of the HF radio until ANVDT reports operational	0	
00:00:01:00 ANVDT_In 33	Returns the state of the radios.		0x11110000 - 286326784
00:00:02:00 ANVDT_In 21	Bad data and nothing will happen		

Table 16. Status Controller Testing

6.5 Audio Multiplexer

The Audio Multiplexer handles the reception input and transmission output respectively from/to the external radios and outputs any audio the audio output channel for the headsets and speaker in real life. In the simulation mode, it will generate a non-zero value for audio and a zero value for no audio.

$$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X = { ((AM_Audio_In, v) | AM_Audio_In ∈ IPorts, v ∈ R),

((HF_Audio_In, v) | HF_Audio_In ∈ IPorts, v ∈ R),

((FM_Audio_In, v) | FM_Audio_In ∈ IPorts, v ∈ R),

((VUHF_Audio_In, v) | VUHF_Audio_In ∈ IPorts, v ∈ R),

((Audio_In, v) | Audio_In ∈ IPorts, v ∈ R) }

Y = { ((AM_Audio_Out, v) | AM_Audio_Out ∈ OPorts, v ∈ R),

((HF_Audio_Out, v) | HF_Audio_Out ∈ OPorts, v ∈ R),

((FM_Audio_Out, v) | FM_Audio_Out ∈ OPorts, v ∈ R),

((VUHF_Audio_Out, v) | VUHF_Audio_Out ∈ OPorts, v ∈ R),

((Audio_Out, v) | Audio_Out ∈ OPorts, v ∈ R) }

S = state ∈ { active, passive}, m_StateOfHf ∈ N, m_StateOfAm ∈ N, m_StateOfFm ∈ N, m_StateOfVuhf ∈ N, m_StateOfAudioIn ∈ N, , m_StateOfAudioOut ∈ N, m_StateOfMic ∈ N, nIndex ∈ N }

δ_{int} = (AM_In | ANVDT_In | FM_In | VUHF_In | Interface_In)
{ passivate(); }

δ_{ext} (s,e,x) = {

```
m_StateOfAudioIn = 0;
//Auditory communications channel all active voice into the speakers and
//headsets, relying on the operators to select and manipulate the conversations
//through the CSU or by brain power!

if( AM_Audio_In == msg.port() )
{
        m_StateOfAm = msg.value();
        if( KEYED == m_StateOfAm )
                m_StateOfAudioIn |= 0x10000028;
        else
                m_StateOfAudioIn |= 0x10000000;
        holdIn( active, Time::Zero );

}

if( FM_Audio_In == msg.port() )
{
        m_StateOfFm = msg.value();
        if( KEYED == m_StateOfFm )
                m_StateOfAudioIn |= 0x01000028;
        else
                m_StateOfAudioIn |= 0x01000000;
        holdIn( active, Time::Zero );

}

if( VUHF_Audio_In == msg.port() )
{
        m_StateOfVuhf = msg.value();
        if( KEYED == m_StateOfVuhf )
                m_StateOfAudioIn |= 0x00100028;
        else
                m_StateOfAudioIn |= 0x00100000;
```

```
                        holdIn( active, Time::Zero );

                }
                if( HF_Audio_In == msg.port() )
                {
                        m_StateOfHf = msg.value();
                        if( KEYED == m_StateOfHf )
                                m_StateOfAudioIn |= 0x00010028;
                        else
                                m_StateOfAudioIn |= 0x00010000;

                        holdIn( active, Time::Zero );

                }
                if( Audio_In == msg.port() )
                {
                        //Determine which radio it is to be outputted on by masking it against
                        //the  encoded radio as declared in the upper byte as per the
                        //Declarations.h file.
                        m_Index = msg.value();

                        //Because the audio multiplexer does not know the state of the radio
        the default is HF

                        //The data set has whether an activate transmitter is selected
                        //Otherwise you can key and talk all you want but you need a
                        //transmitter active not just selected.
                        if(m_Index & 0x100 && 65536 < m_Index )
                                m_StateOfAudio = AUDIO;
                        else m_StateOfAudio = NO_AUDIO;

                        //Hot mic keyed transmitter
                        if(m_Index & 0x1 && 65536 < m_Index )
                                m_StateOfMic = KEYED;
                        else m_StateOfMic = UNKEYED;

                        //Now we have the index for the radio and a flag for audio transmission
                        holdIn( active, Time::Zero );
                }
        }
λ (S) = {
        //Transmission mode
        if( KEYED == m_StateOfMic)
        {
                if( 0x11110000 & m_Index )
                {
                        sendOutput( msg.time(),HF_Audio_Out, m_StateOfAudio );
```

```
            sendOutput( msg.time(),VUHF_Audio_Out, m_StateOfAudio );
            sendOutput( msg.time(),FM_Audio_Out, m_StateOfAudio );
            sendOutput( msg.time(),AM_Audio_Out, m_StateOfAudio );
        }
        else
        {
                if( 0x00010000 & m_Index )
            {
                sendOutput( msg.time(), HF_Audio_Out, m_StateOfAudio );
            }
            if( 0x00100000 & m_Index )
            {
                sendOutput( msg.time(),VUHF_Audio_Out, m_StateOfAudio );
            }
            if(  0x01000000 & m_Index)
            {
                sendOutput( msg.time(), FM_Audio_Out, m_StateOfAudio );
            }
            if( 0x10000000 & m_Index )
            {
                sendOutput( msg.time(), AM_Audio_Out, m_StateOfAudio );
            }
        }
        m_StateOfAudio = 0;

        }

        //Now produce a valid audio output for the speaker and the final output
        if( 65536 < m_StateOfAudioIn)
        {
            sendOutput( msg.time(), Audio_Out, m_StateOfAudioIn );
            //reset the audio
            m_StateOfAudioIn = 0;
        }
    }
```

6.5.1 Audio Multiplexer Testing

Testing of this atomic model can accomplished by having one or more discrete events inputted and examining the output.

Thus with an input as detailed in the following table, an output would be generated as follows:

Input	Remarks	Output port (Hexadecimal – Decimal)

00:00:01:00 Am_In 40	AM Transceiver reception message	0x1000028 – 268435496 Audio_Out			
00:00:02:00 Fm_In 40	FM Transceiver reception message	0x01000028 – 16777256 Audio_Out			
00:00:03:00 Vuhf_In 40	VUHF Transceiver reception message	0x00100028 – 1048616 Audio_Out			
00:00:04:00 HF_In 40	HF reception message	0x00010028 – 65576 Audio_Out			
00:00:05:00 Audio_In 0	No active transmitters, no key, no audio, txselected				
00:00:06:00 Audio_In 65809	Keyline, HF transmitter selected, audio	34 HF_Audio_Out			
00:00:07:00 Audio_In 1048849	Keyline, VUHF transmitter selected, audio		34 VUHF_Audio_Out		
00:00:08:00 Audio_In 16777489	Keyline, FM transmitter selected, audio f			34 FM_Audio_out	
00:00:09:00 Audio_In 268435729	Keyline, AM transmitter selected, audio				34 AM_Audio_Out
00:00:10:00 Audio_In 286327057	Keyline, transmitter selected, audio for all radios	34 HF_Audio_Out	34 VUHF_Audio_Out	34 FM_Audio_out	34 AM_Audio_Out

Table 17. Audio Multiplexer Testing

7. Exclusions

7.2 VUHF Receiver-Transmitter

The AN/ARC-210(V) Multimode radio provides a jam-resistant, two-way, voice and data communication radio for the tactical aircraft environment over the frequency range of 30 to 512 MHz. It can operate in normal, secure or jam-resistant modes. For the purposes of simplicity of this model, this item is not modelled.

7.3 AM Receiver-Transmitter

The AN/ARC-511 radio set is primarily used by the pilot and copilot to communicate with Air Traffic Control (ATC) facilities and can also be used for direction finding (DF) or homing which will not be modelled. For the purposes of simplicity of this model, this item is not modelled.

7.4 FM Receiver-Transmitter

The ARC-513(V2) radio allows voice communications with government agencies, maritime mobile units and the simultaneous monitoring of the VHF maritime mobile emergency frequency (156.8 MHz). The radio is also used for direction finding (DF) or homing. The VHF-FM radio can be used to provide aural monitoring of sonobuoy signals plus DF or homing to a particular sonobuoy. For the purposes of simplicity of this model, this item is not modelled.

7.5 LED

The amount of LEDs in the real-system is not conducive to the amount of effort for this assignment. Thus, only one LED is modelled.

7.6 CSU

The 12 CSU in the real-system is not conducive to the amount of effort for this assignment. Thus, only one CSU is modelled.

7.7 CCU

The dual CCU in the real-system is not conducive to the amount of effort for this assignment. Thus, only one CCU is modelled.

7.8 Radios

The four VUHF secure radios and the dual HF transceiver interconnected with the singular ANVDT in the real-system is not conducive to the amount of effort for this assignment. Thus, only one ANVDT and HF transceiver is modelled. The HF control head was not modelled also as per Figure 4 because of security restrictions on its operation.

8 State Diagrams

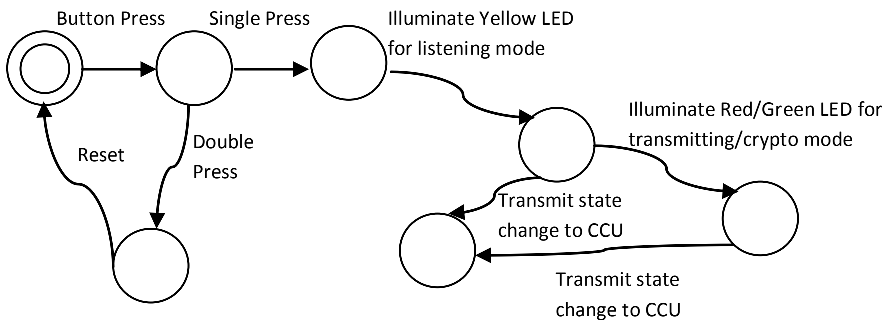

Figure 9. CSU State Diagram - CSU User Selection

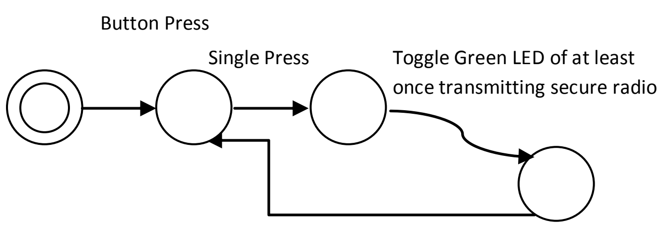

Figure 10. Crypto State Diagram - Crypto User Selection

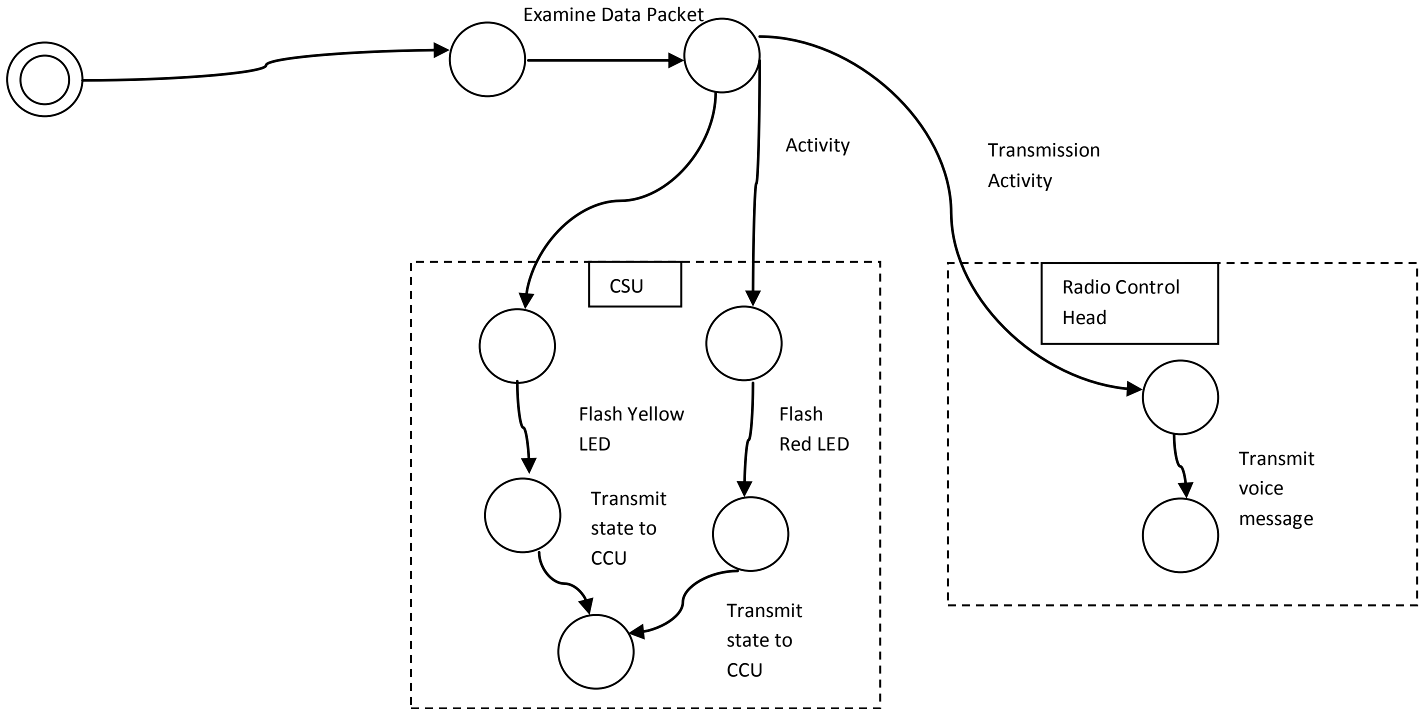

Figure 11. CMS Transmission Radio Activity State Diagram

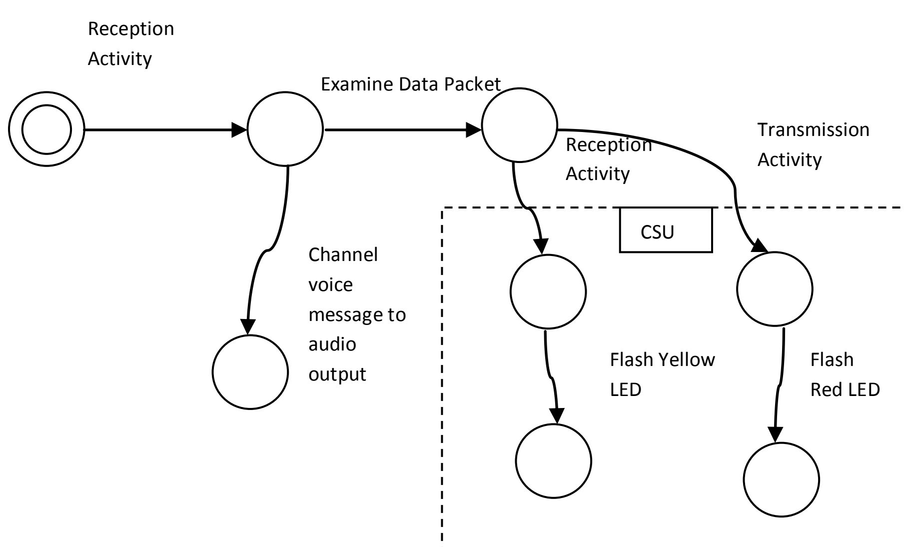

Figure 12. CMS Reception Radio Activity State Diagram

9 Results

The model was coded, commencing with the atomic models driving the inputs and outputs in order to validate the outputs. Upon success, other joint atomic models were added into the coupled versions of the models. Finally the couple models were joined together into one large model and its integration performance was examined.

Though appearing to be the best method for testing using the vertical stovepipe approach to regression testing, the dissimilarities in the models from day to day implementation and the complexity of the model as a whole required massive amounts of regression testing in order to produce a working model.

As of to this date, the model appears fully tested, though as with any computer application the coverage of the testing is only as good as the implementation of the testing. Significant obstructions were discovered in the testing such as the lack of a debugger, a memory watch or even a variable watch, and the limit options available in the CD++ perspective compared to a C++ or even java perspective on Eclipse.

The atomic model would not accept parameters data such as preparation time or processing time causing the programmer to control the operation through the timing of the inputs. This flaw greatly increased the development time because of the requirement to understand, remember and implement the dataflow structure of the model at all times.

Significant resources were underutilized because of the unfamiliarity of the atomic model modeller. This output graphic renders the value and states based on the log file created during the simulation exercise. The tool is designed around limited value variety and was not useful in examining a byte packed integer. As a lot of simulators use the q16, q32, or even binary coded decimals format, this is a limitation that is of concern as the modelled output could be significantly different from the real entity unless a translator mechanism was implemented.

However, the model was coded and operational. Data was recorded in the log files and out files and used to generate the following figures.

These figures demonstrate the successful operation, through the selection of user inputs to generate lighting logic and auditory output.

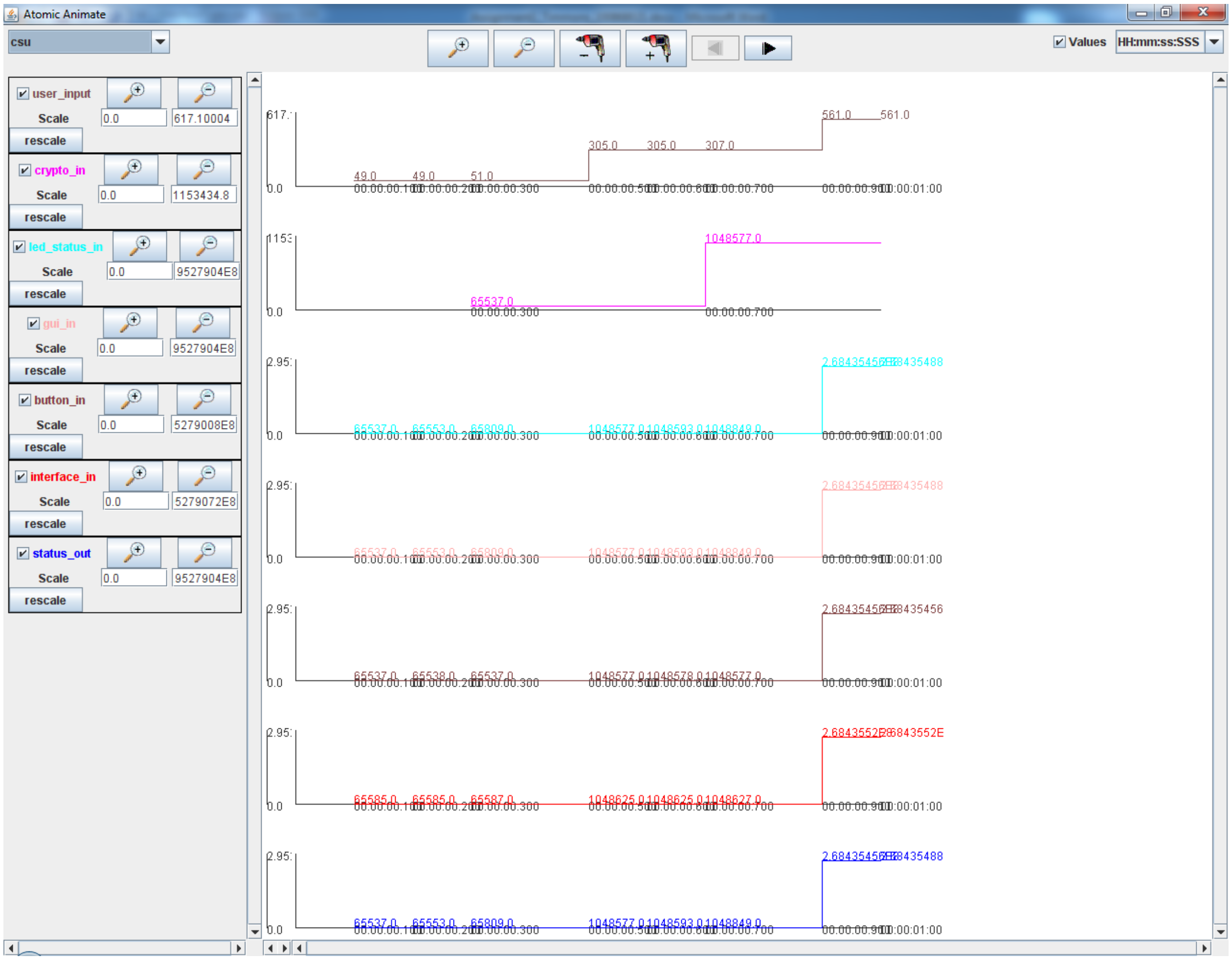

Figure 13. Atomic Modeling of the CSU Operation

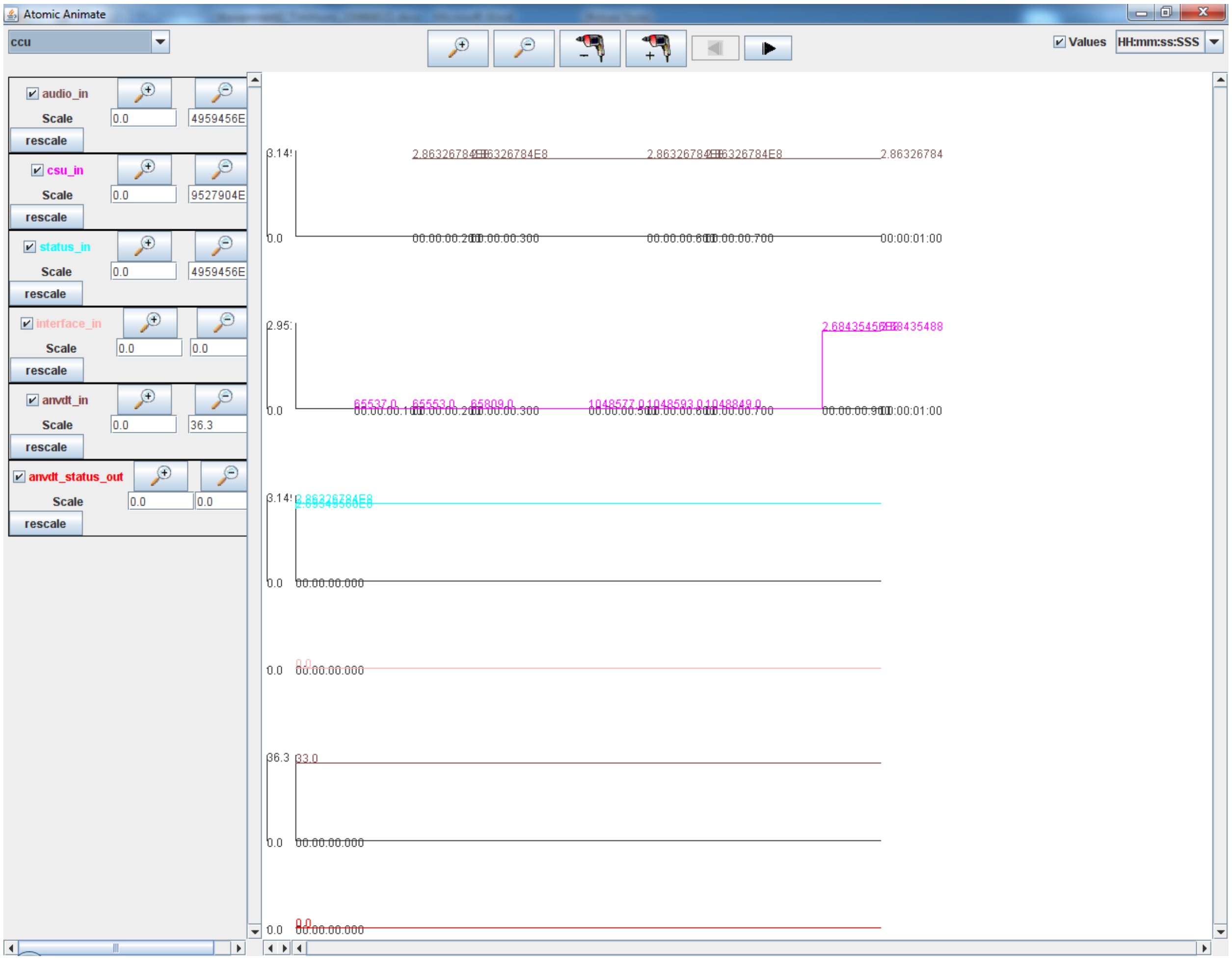

Figure 14. Atomic Modeling of the CCU Operation

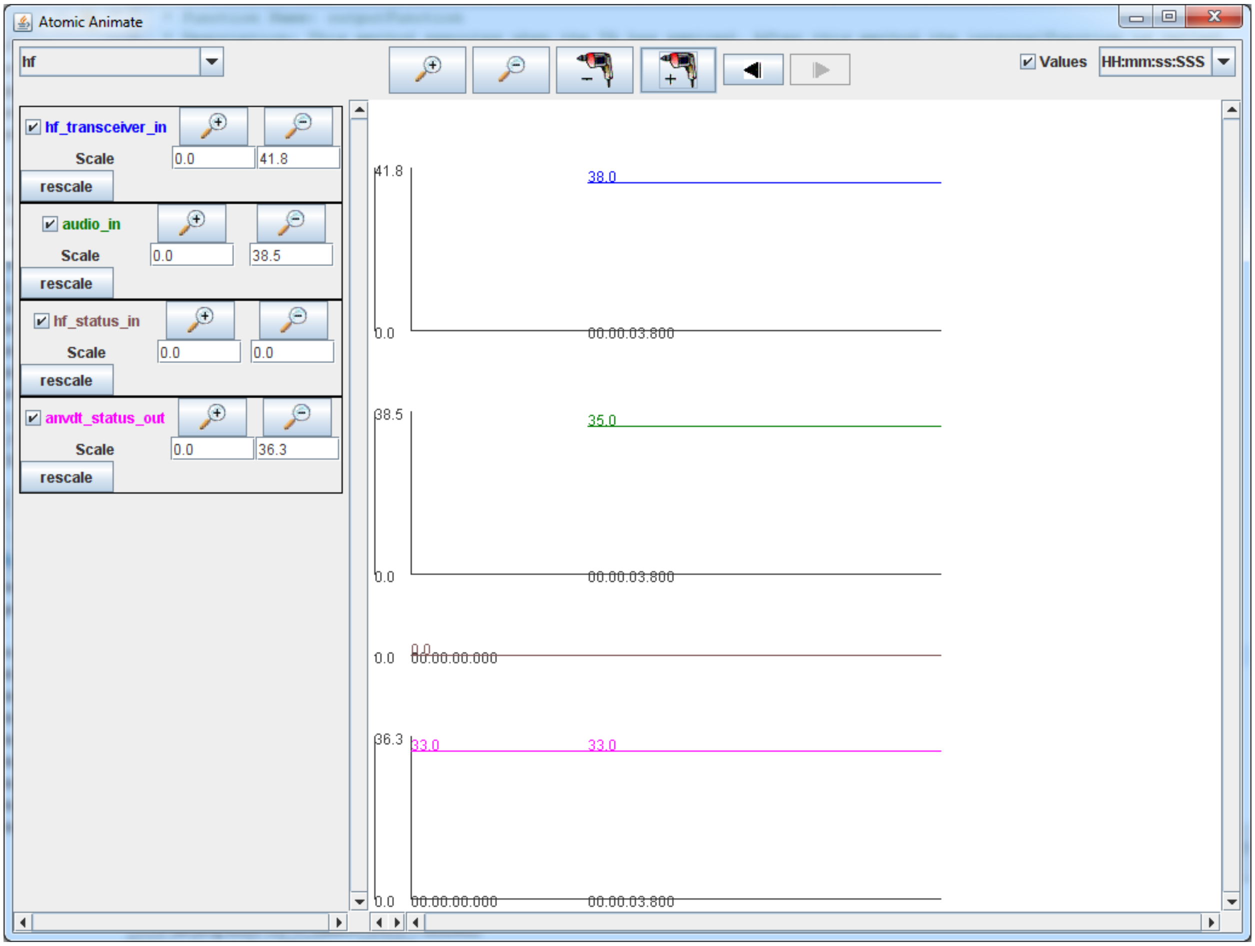

Figure 15. Atomic Modeling of the HF Operation