1. Primary user Engine

1.1 Markov Chain

A Markov process is a random process for which the future (the next step) depends only on the present state; it has no memory of how the present state was reached. We can describe a Markov chain as follows: We have a set of states, S = {s1, s2, . . ., sr}. The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state si, then it moves to state sj at the next step with a probability denoted by pij, and this probability does not depend upon which states the chain was in before the current state where Markov model is memoryless system. The probabilities pij are called transition probabilities. The process can remain in the state it is in, and this occurs with probability pi. An initial probability distribution, defined on S, specifies the starting state. Usually this is done by specifying a particular state as the starting state.

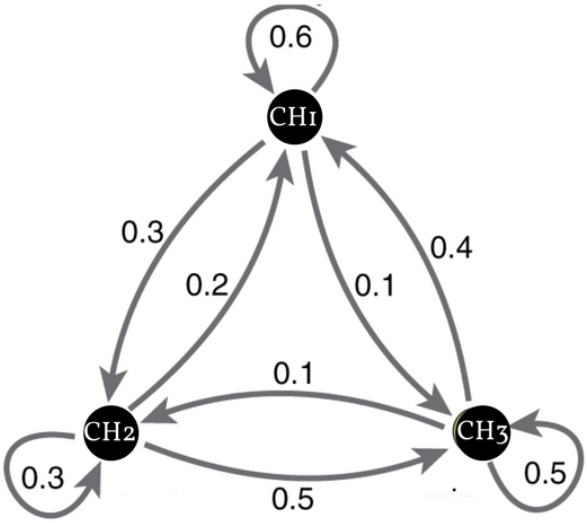

Figure 1-1 Markov Chain of PU states transition

The engine of primary user is designed to operate in a random manner, to switch its operation center frequency from channel to another according the Markov model as shown in (figure 1.1) and We initialize the data produces the following transition probabilities as shown in table below.

MARKOV Chain Probability			
states transition			
	CHANNEL 1.	**CHANNEL 2**	**CHANNEL 3**
CHANNEL 1	P(CH_1|CH_1)= 0.1	P(CH_1|CH_2)=0.3	P(CH_1|CH_3)=0.6
CHANNEL 2	P(CH_2|CH_1)= 0.1	P(CH_2|CH_2)=0.5	P(CH_2|CH_3)=0.4
CHANNEL 3	P(CH_3|CH_1)= 0.1	P(CH_3|CH_2)=0.2	P(CH_3|CH_3)=0.7

1.2 Algorithm Implementation:

RANDOM OUTCOME FUNCTION

```
void CE_PU_Markov_Chain_VER3::RANDOM_OUTOCME(ExtensibleCognitiveRadio *ECR) {
static constexpr float Sample_Space[10]={0,1,2,3,4,5,6,7,8,9};
outcome= rand() % 10;
state_probability = Sample_Space[outcome];
}
```

STATE TRANSITION FOR TX FUNCTION

```
void CE_PU_Markov_Chain::PU_TX_Behaviour(ExtensibleCognitiveRadio *ECR) {
get the value of TX_Frequency;

//checking for first state CHANNEL_1
if (tx_freq == CHANNEL_1){
if (state_probability == 0)
ECR->set_tx_freq(CHANNEL_1);
else if(state_probability>=1 || state_probability<4)
ECR->set_tx_freq(CHANNEL_2);
else
ECR->set_tx_freq(CHANNEL_3);
}

Checking for CHANNEL_2 AND CHANNEL_3 according the transition matrix
}
end
```

1.2.1 Output Samples from Terminal:

- +--
  HOPPING # 1
- +--
  Current State: CHANNEL 1 ::::: Next State: CHANNEL 1
- +--
  HOPPING # 2
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 3
- +--
  Current State: CHANNEL 1 ::::: Next State: CHANNEL 1
- +--
  HOPPING # 4
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 5
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 6
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 7
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 8
- +--
  Current State: CHANNEL 2 ::::: Next State: CHANNEL 2
- +--
  HOPPING # 9
- +--

1.2.2 Output Samples Using spectrum analyzer node:

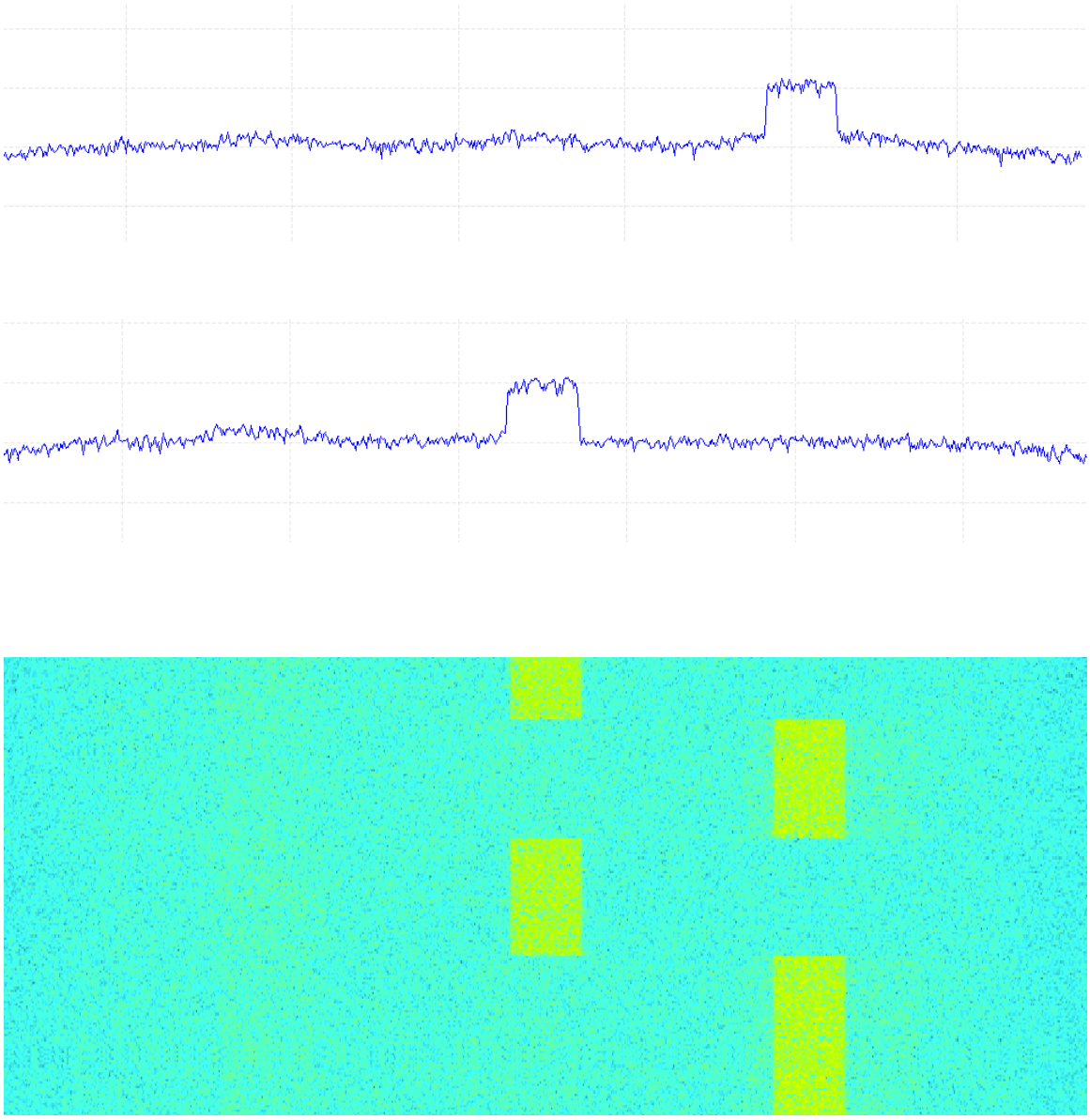

Figure 1-2 PU Activity in 800MHz Band