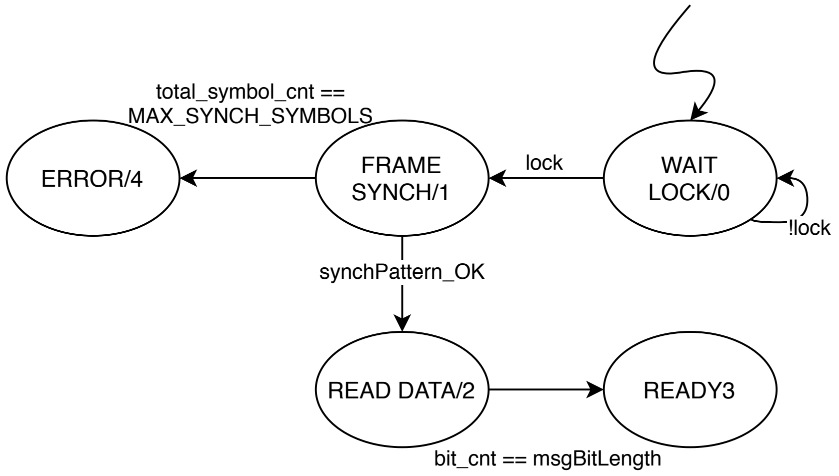
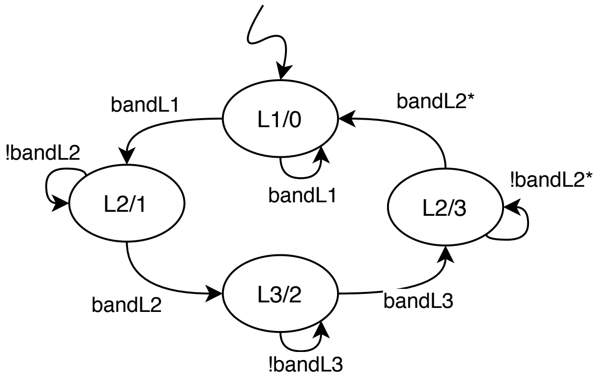
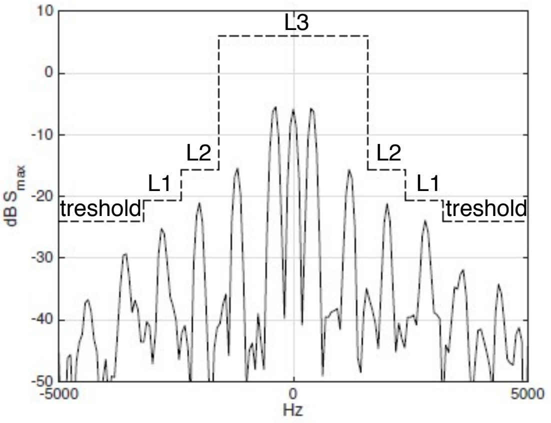
FSM
Decoder

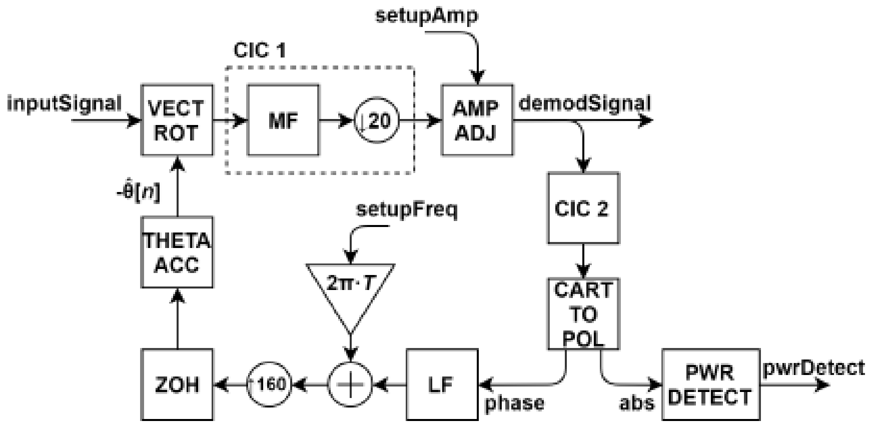


FSM
Calc Mask





pttA2Demod
Block
Diagram



ALGORITHM 1

Decoder TestBench

```
Require: inputSignal
  loop
    scanf(inputSignal)                                          ▷ Load input signal
  end loop
  for decoder_idx = 0 : NUMBER_OF_DECODERS do
    call clearDecoder(decoder_idx)                              ▷ Clear all decoder channels
  end for
  loop
    for $iSmp = 0 : WindowLength$ do
      $windowedSignal \leftarrow inputSignal(iSmp + WindowLength * nW)$    ▷ Slice the input signal in the window length nW times
    end for
    call UpdateTimeout                                          ▷ Clear the channel where processed windows overflow
    call Detect_Loop(windowedSignal)                            ▷ Detect Loop returns new detected PTTs Freq_amp and Freq_idx
    if New Detected PTTs then                                   ▷ Assignment of parameters to decod PTT signal
      vga ← vgaGain(Freq_amp)                                   ▷ call vgaGain to calculate Mantissa and Exponente
      initFreq ← Freq_idx
      FSM.state ← FRAME_SYNCH
      Update active_list channels
    end if
    for decoder_idx = 0 : NUMBER_OF_DECODERS do
      if Channel.isActive then
        call Demod_Loop(windowedSignal,vga,initFreq)            ▷ The Demod Loop returns arrays with 8 positions each symbLock and symbOut
        for $iSymb = 0 : nSymb$ do
          if $symbLock(iSymb)$ then
            if FSM.state == FRAME_SYNCH then
              call frameSynch(symbOut)                          ▷ Decode and compare the output with the last 10 bits of the sync word pattern
            else if FSM.state == DATA then
              call readData(symbOut)                            ▷ Decode and store the output
              if FSM.state == READY then
                Show result
              end if
            else if FSM.state == ERROR then
              call clearDecoder(decoder_idx)
            end if
          end if
        end for
      end if
    end for
  end loop
```

□