1- To cope with the requested design, we should modify the stepper_motor_clkpulse module.

Firstly, we should add a new input argument called *speed_select*.

Secondly, we need a register and its corresponding state machine to save the motor's current speed.

And finally, we should modify the step motor FSM to check the motor speed and then select the pulse period correctly.

The following code shows the solution.

```
#define PULSE_PERIOD_100HZ 1000000
#define PULSE_PERIOD_50HZ  2000000
#define PULSE_PERIOD_25HZ  4000000

typedef enum{slow, medium, fast} speed_type;

typedef enum{zero, one} states_type;
void stepper_motor_clkpulse(bool speed_select, bool &clkpulse) {
#pragma HLS INTERFACE ap_none port=clkpulse
#pragma HLS INTERFACE ap_none port=speed_select
#pragma HLS INTERFACE ap_ctrl_none port=return


    static speed_type speed_state = slow;
    speed_type        next_speed_state = speed_state;

    switch (speed_state) {
    case slow:
        if (speed_select == 1) {
            next_speed_state = medium;
        } else {
            next_speed_state = slow;
        }
        break;
    case medium:
        if (speed_select == 1) {
            next_speed_state = fast;
        } else {
            next_speed_state = medium;
        }
```

```
        break;
    case fast:
        if (speed_select == 1) {
            next_speed_state = slow;
        } else {
            next_speed_state = fast;
        }
        break;
    default:
        break;
    }

    static states_type state    = zero;
    static unsigned int counter = PULSE_PERIOD_25HZ-1;


    states_type next_state;
    unsigned int          next_counter;

    bool clkpulse_local;

    switch(state) {
    case zero:
        if (counter == 0) {
            if (speed_state == slow) {
                next_counter           = PULSE_PERIOD_25HZ-1;
            } else if (speed_state == medium) {
                next_counter           = PULSE_PERIOD_50HZ-1;
            } else {
                next_counter           = PULSE_PERIOD_100HZ-1;
            }
            next_state             = one;
            clkpulse_local         = 1;
        } else {
            next_counter           = counter-1;
            next_state             = zero;
            clkpulse_local         = 0;
        }
        break;
    case one:
        next_counter           = counter-1;
        next_state             = zero;
        clkpulse_local         = 0;
```

```
            break;
      default:
            break;
      }

      speed_state = next_speed_state;
      state = next_state;
      counter = next_counter;
      clkpulse = clkpulse_local;
}
```

2- To solve the requested problem, we should make the following changes to the original stepper motor driver code.
   a. Add a new input argument named *direction*.
   b. Add a register and the corresponding FSM to keep track of changes in the direction
   c. In the main FSM, each state should check the direction and then change the state appropriately.

```
typedef enum {clockwise, anticlockwise} stepper_motor_direction;

typedef enum {s0, s1, s2, s3, s4, s5, s6, s7} stepper_motor_state;

void stepper_motor_driver ( bool direction, bool motor_clk_rate,
ap_uint<4> &motor_signal) {
#pragma HLS INTERFACE ap_none port=direction
#pragma HLS INTERFACE ap_none port=motor_signal
#pragma HLS INTERFACE ap_none port=motor_clk_rate
#pragma HLS INTERFACE ap_ctrl_none port=return

      static stepper_motor_direction direction_state = clockwise;
      stepper_motor_direction next_direction_state =
direction_state;

      static stepper_motor_state state=s0;

      switch (direction_state) {
      case clockwise:
            if (direction == 1)
                  next_direction_state = anticlockwise;
            else
                  next_direction_state = clockwise;
            break;
      case anticlockwise:
            if (direction == 1)
                  next_direction_state = clockwise;
            else
                  next_direction_state = anticlockwise;
            break;
      default:
            break;

      }
      stepper_motor_state next_state=state;
```

```
    ap_uint<4> motor_signal_tmp;

    switch (state) {
        case s0:
            if (motor_clk_rate == 1) {
                if (direction_state == clockwise)
                    next_state = s1;
                else
                    next_state = s7;
            } else {
                next_state = s0;
            }
            motor_signal_tmp = 0b1000;
            break;
        case s1:
            if (motor_clk_rate == 1) {
                if (direction_state == clockwise)
                    next_state = s2;
                else
                    next_state = s0;
            } else {
                next_state = s1;
            }
            motor_signal_tmp = 0b1001;

            break;
        case s2:
            if (motor_clk_rate == 1) {
                if (direction_state == clockwise)
                    next_state = s3;
                else
                    next_state = s1;
            } else {
                next_state = s2;
            }
            motor_signal_tmp = 0b0001;

            break;
        case s3:
            if (motor_clk_rate == 1) {
                if (direction_state == clockwise)
                    next_state = s4;
                else
                    next_state = s2;
            } else {
                next_state = s3;
```

```
            }
            motor_signal_tmp = 0b0011;

            break;
      case s4:
            if (motor_clk_rate == 1) {
                  if (direction_state == clockwise)
                        next_state = s5;
                  else
                        next_state = s3;
            } else {
                  next_state = s4;
            }
            motor_signal_tmp = 0b0010;

            break;
      case s5:
            if (motor_clk_rate == 1) {
                  if (direction_state == clockwise)
                        next_state = s6;
                  else
                        next_state = s4;
            } else {
                  next_state = s5;
            }
            motor_signal_tmp = 0b0110;

            break;
      case s6:
            if (motor_clk_rate == 1) {
                  if (direction_state == clockwise)
                        next_state = s7;
                  else
                        next_state = s5;
            } else {
                  next_state = s6;
            }
            motor_signal_tmp = 0b0100;

            break;
      case s7:
            if (motor_clk_rate == 1) {
                  if (direction_state == clockwise)
                        next_state = s0;
                  else
                        next_state = s6;
```

```
                } else {
                      next_state = s7;
                }
                motor_signal_tmp = 0b1100;
                break;

          default:
                break;
     }

     direction_state = next_direction_state;
     state = next_state;
     motor_signal = motor_signal_tmp;
}
```