```c
/*
 * File:   stepperDriver.c
 * Author: ricch
 *
 * Created on August 30, 2023, 10:39 PM
 *
 * DRV8432 driver 2H bridge
 */


#include <stepperDriver.h>

static STEPPER_DATA stepperData;
extern APP_DATA appData;


//------------------------------------------------------------------------------//
initStepperData
void initStepperParam(void){

    stepperData.isAtHomeInCW     = false;
    stepperData.isAtHomeInCCW    = false;
    stepperData.isIndexed        = false;
    stepperData.isInAutoHomeSeq  = false;

    stepperData.performedSteps   = 0;
    stepperData.stepToReach      = 0;

    stepperData.stepPerSec       = 1000;

    stepperData.stepPerTurn      = 200;
    stepperData.gearValue        = 200;

    stepperData.anglePerStep     = 1.8;

    stepperData.dutyCycleStepper = 30;
}

void initStepperMotor(){

    //setStepperPower(&stepperData, &stepperData.dutyCycleStepper);

    /* Disable RESET on both H bridge */
    RESET_AB_CMDOn();
    RESET_CD_CMDOn();
}

//------------------------------------------------------------------------------//
turnOffStepperPwms
/* Disable all PWMs for motor control */
void turnOffStepperPwms(void){

    /* A */
    PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
    /* B */
    PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
    /* A_ */
    PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
    /* B_ */
    PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
}

//------------------------------------------------------------------------------//
changeSpeed
void changeSpeed(STEPPER_DATA *pStepperData){

    uint16_t tmrPerdiod = 0;
    uint16_t frequency = 0;
    //uint16_t presc = 0;
```

```
67
68      frequency = pStepperData->stepPerSec;
69      //presc = TMR_PrescaleGet_Default(TMR_ID_3);
70      tmrPerdiod = SYS_CLK / (frequency * 16) - 1;
71      PLIB_TMR_Counter16BitClear(TMR_ID_3);
72      PLIB_TMR_Period16BitSet(TMR_ID_3, tmrPerdiod);
73  }
74
75  //------------------------------------------------------------------------//
    processStepper
76  void processStepper(STEPPER_DATA *pStepperData){
77
78      static uint8_t step = 0;
79      //---------------------------// Counter clockwise CCW
80      if(pStepperData->performedSteps > pStepperData->stepToReach){
81          if(pStepperData->isAtHomeInCCW == false){
82              switch(step){
83                  /* Sequence of 4 steps for CCW rotation */
84                  case 1:
85                      /* A */
86                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
87                      /* B */
88                      PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
89                      /* A_ */
90                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
91                      /* B_ */
92                      PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
93                      break;
94
95                  case 2:
96                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
97                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
98                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
99                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
100                     break;
101
102                 case 3:
103                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
104                     PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
105                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
106                     PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
107                     break;
108
109                 case 0:
110                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
111                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
112                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
113                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
114                     break;
115             }
116             step++;
117         }
118         /* Four steps performed in CCW */
119         if(step == 4){
120
121             step = 0;
122             pStepperData->performedSteps -= 4;
123         }
124         /* Index is reach in CCW */
125         if(INDEXStateGet() && pStepperData->isAtHomeInCW == false){
126
127             pStepperData->isAtHomeInCCW = true;
128 //            pStepperData->stepToDoReach = pStepperData->performedStep;
129
130             if(pStepperData->isInAutoHomeSeq == true){
131
132                 pStepperData->stepToReach = 0;
133                 pStepperData->performedSteps = 0;
134                 pStepperData->isIndexed = true;
```

```
                    pStepperData->isInAutoHomeSeq = false;
                }
            }
            else pStepperData->isAtHomeInCCW = false;
        }
        //---------------------------// Clockwise CW
        else if(pStepperData->performedSteps < pStepperData->stepToReach){
            if(pStepperData->isAtHomeInCW == false){
                switch(step){
                    /* Sequence of 4 steps for CW rotation */
                    case 1:
                        /* A */
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        /* B */
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        /* A_ */
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        /* B_ */
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 0:
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 3:
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 2:
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;
                }
                step++;
            }
            /* Four steps performed in CW */
            if(step == 4){

                step = 0;
                pStepperData->performedSteps += 4;
            }
            /* Index is reach in CW */
            if(INDEXStateGet() && pStepperData->isAtHomeInCCW == false){

                pStepperData->isAtHomeInCW = true;
                /* Stop the automatic sequence */
                appData.isFullImaginSeqEnable = false;
                /* Stop the motor */
                pStepperData->stepToReach = pStepperData->performedSteps;
            }
            else pStepperData->isAtHomeInCW = false;
        }


        // The motor reach its desired position
    //    if(pStepperData->performedSteps == pStepperData->stepToReach){
    ////        turnOffStepperPwms();
    //    } else {
    //
    ////        PLIB_MCPWM_Enable(MCPWM_ID_0);
```

```
204   //    }
205   }
206
207
208
209   //-------------------------------------------------------------------------// setSpeed
210   void setSpeed(STEPPER_DATA *pStepperData, uint32_t *pStepPerSec){
211
212       // Limit values to avoid problems
213       if(*pStepPerSec < STEP_PER_SEC_MIN) *pStepPerSec = STEP_PER_SEC_MIN;
214       if(*pStepPerSec > STEP_PER_SEC_MAX) *pStepPerSec = STEP_PER_SEC_MAX;
215
216       // Save data
217       pStepperData->stepPerSec = *pStepPerSec;
218   }
219
220   int32_t getSpeed(STEPPER_DATA *pStepperData){
221
222       return pStepperData->stepPerSec;
223   }
224
225   //-------------------------------------------------------------------------//
      setGearReduction
226   void setGearReduction(STEPPER_DATA *pStepperData, uint32_t *pGearValue){
227
228       // Limit values to avoid problems
229       if(*pGearValue < GEAR_VALUE_MIN) *pGearValue = GEAR_VALUE_MIN;
230       if(*pGearValue > GEAR_VALUE_MAX) *pGearValue = GEAR_VALUE_MAX;
231
232       // Save data
233       pStepperData->gearValue = *pGearValue;
234   }
235   //-------------------------------------------------------------------------//
      getGearReduction
236   uint32_t getGearReduction(STEPPER_DATA *pStepperData){
237
238       return pStepperData->gearValue;
239   }
240
241   //-------------------------------------------------------------------------//
      setAnglePerStep
242   void setAnglePerStep(STEPPER_DATA *pStepperData, uint32_t *pAnglePerStep){
243
244       float temp = (*pAnglePerStep / 10.0);
245
246       // Limit values to avoid problems
247       if(temp < ANGLE_PER_STEP_MIN) temp = (ANGLE_PER_STEP_MIN);
248       if(temp > ANGLE_PER_STEP_MAX) temp = (ANGLE_PER_STEP_MAX);
249       *pAnglePerStep = temp * 10;
250
251       // Save data
252       pStepperData->anglePerStep = temp;
253   }
254   //-------------------------------------------------------------------------//
      getAnglePerStep
255   uint32_t getAnglePerStep(STEPPER_DATA *pStepperData){
256
257       // x10 ???
258       return pStepperData->anglePerStep * 10;
259   }
260
261   //-------------------------------------------------------------------------//
      getPerformedSteps
262   int32_t getPerformedSteps(STEPPER_DATA *pStepperData){
263
264       return pStepperData->performedSteps / pStepperData->stepPerTurn;
265   }
266
267
```

```
268 //------------------------------------------------------------------------//
    setRotationToDo
269 void setRotationToDo(STEPPER_DATA *pStepperData, int32_t *pRotationToDo){
270
271     // Limit values to avoid problems
272     if(*pRotationToDo < ROTATION_TO_DO_MIN) *pRotationToDo = ROTATION_TO_DO_MIN;
273     if(*pRotationToDo > ROTATION_TO_DO_MAX) *pRotationToDo = ROTATION_TO_DO_MAX;
274
275     // Save data
276     pStepperData->stepToReach = *pRotationToDo * pStepperData->stepPerTurn;
277 }
278 //------------------------------------------------------------------------//
    getRotationTodo
279 int32_t getRotationToDo(STEPPER_DATA *pStepperData){
280
281     return pStepperData->stepToReach / pStepperData->stepPerTurn;
282 }
283
284 //------------------------------------------------------------------------// autoHome
285 void startAutoHome(STEPPER_DATA *pStepperData){
286
287     pStepperData->isInAutoHomeSeq = true;
288     // Check if the arm is not at home
289     if(pStepperData->isAtHomeInCCW == false){
290         // Put steps to do for returning home in CCW
291         pStepperData->stepToReach = -50000; // DEFINE? STEP_TO_DO_MAX
292     }
293 }
294
295 //------------------------------------------------------------------------//
    setStepperPower
296 void setStepperPower(STEPPER_DATA *pStepperData, uint16_t *pDutyCycleStepper){
297
298     uint16_t dutyValCh1 = 0;
299
300     // Limit values to avoid problems
301     if(*pDutyCycleStepper < MCPWM_DUTYCYCLE_MIN) *pDutyCycleStepper
302             = MCPWM_DUTYCYCLE_MIN;
303     if(*pDutyCycleStepper > MCPWM_DUTYCYCLE_MAX) *pDutyCycleStepper
304             = MCPWM_DUTYCYCLE_MAX;
305
306     /* Save configuration in the structure */
307     pStepperData->dutyCycleStepper = *pDutyCycleStepper;
308
309     /* Must be the inverse of the CHANNEL 1 */
310     dutyValCh1 = MCPWM_PRIMARY_PERIOD - *pDutyCycleStepper;
311
312     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL1,
313             dutyValCh1);
314     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL2,
315             *pDutyCycleStepper);
316 }
317
318 int16_t getStepperPower(STEPPER_DATA *pStepperData){
319
320     return pStepperData->dutyCycleStepper;
321 }
322
323
324 //------------------------------------------------------------------------//
    getStepperStruct
325 STEPPER_DATA* getMyStepperStruct(void){
326
327     /* Return the address of the structure */
328     return &stepperData;
329 }
```