```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;

    frequency = pStepperData->stepPerSec;
    //presc = TMR_PrescaleGet_Default(TMR_ID_3);
    tmrPerdiod = SYS_CLK / (frequency * 16) - 1;
    PLIB_TMR_Counter16BitClear(TMR_ID_3);
    PLIB_TMR_Period16BitSet(TMR_ID_3, tmrPerdiod);
}

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

    static uint8_t step = 0;
    //---------------------------// Counter clockwise CCW
    if(pStepperData->performedSteps > pStepperData->stepToReach){
        if(pStepperData->isAtHomeInCCW == false){
            switch(step){
                /* Sequence of 4 steps for CCW 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 2:
                    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 0:
                    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 CCW */
        if(step == 4){

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

            pStepperData->isAtHomeInCCW = true;
//            pStepperData->stepToDoReach = pStepperData->performedStep;

            if(pStepperData->isInAutoHomeSeq == true){

                pStepperData->stepToReach = 0;
                pStepperData->performedSteps = 0;
                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);
```

```
//    }
}



//----------------------------------------------------------------------------// setSpeed
void setSpeed(STEPPER_DATA *pStepperData, uint32_t *pStepPerSec){

    // Limit values to avoid problems
    if(*pStepPerSec < STEP_PER_SEC_MIN) *pStepPerSec = STEP_PER_SEC_MIN;
    if(*pStepPerSec > STEP_PER_SEC_MAX) *pStepPerSec = STEP_PER_SEC_MAX;

    // Save data
    pStepperData->stepPerSec = *pStepPerSec;
}

int32_t getSpeed(STEPPER_DATA *pStepperData){

    return pStepperData->stepPerSec;
}

//----------------------------------------------------------------------------// setGearReduction
void setGearReduction(STEPPER_DATA *pStepperData, uint32_t *pGearValue){

    // Limit values to avoid problems
    if(*pGearValue < GEAR_VALUE_MIN) *pGearValue = GEAR_VALUE_MIN;
    if(*pGearValue > GEAR_VALUE_MAX) *pGearValue = GEAR_VALUE_MAX;

    // Save data
    pStepperData->gearValue = *pGearValue;
}
//----------------------------------------------------------------------------// getGearReduction
uint32_t getGearReduction(STEPPER_DATA *pStepperData){

    return pStepperData->gearValue;
}

//----------------------------------------------------------------------------// setAnglePerStep
void setAnglePerStep(STEPPER_DATA *pStepperData, uint32_t *pAnglePerStep){

    float temp = (*pAnglePerStep / 10.0);

    // Limit values to avoid problems
    if(temp < ANGLE_PER_STEP_MIN) temp = (ANGLE_PER_STEP_MIN);
    if(temp > ANGLE_PER_STEP_MAX) temp = (ANGLE_PER_STEP_MAX);
    *pAnglePerStep = temp * 10;

    // Save data
    pStepperData->anglePerStep = temp;
}
//----------------------------------------------------------------------------// getAnglePerStep
uint32_t getAnglePerStep(STEPPER_DATA *pStepperData){

    // x10 ???
    return pStepperData->anglePerStep * 10;
}

//----------------------------------------------------------------------------// getPerformedSteps
int32_t getPerformedSteps(STEPPER_DATA *pStepperData){

    return pStepperData->performedSteps / pStepperData->stepPerTurn;
}


```

```c
//----------------------------------------------------------------------------// setRotationToDo
void setRotationToDo(STEPPER_DATA *pStepperData, int32_t *pRotationToDo){

    // Limit values to avoid problems
    if(*pRotationToDo < ROTATION_TO_DO_MIN) *pRotationToDo = ROTATION_TO_DO_MIN;
    if(*pRotationToDo > ROTATION_TO_DO_MAX) *pRotationToDo = ROTATION_TO_DO_MAX;

    // Save data
    pStepperData->stepToReach = *pRotationToDo * pStepperData->stepPerTurn;
}
//----------------------------------------------------------------------------// getRotationTodo
int32_t getRotationToDo(STEPPER_DATA *pStepperData){

    return pStepperData->stepToReach / pStepperData->stepPerTurn;
}

//----------------------------------------------------------------------------// autoHome
void startAutoHome(STEPPER_DATA *pStepperData){

    pStepperData->isInAutoHomeSeq = true;
    // Check if the arm is not at home
    if(pStepperData->isAtHomeInCCW == false){
        // Put steps to do for returning home in CCW
        pStepperData->stepToReach = -50000; // DEFINE? STEP_TO_DO_MAX
    }
}

//----------------------------------------------------------------------------// setStepperPower
void setStepperPower(STEPPER_DATA *pStepperData, uint16_t *pDutyCycleStepper){

    uint16_t dutyValCh1 = 0;

    // Limit values to avoid problems
    if(*pDutyCycleStepper < MCPWM_DUTYCYCLE_MIN) *pDutyCycleStepper
            = MCPWM_DUTYCYCLE_MIN;
    if(*pDutyCycleStepper > MCPWM_DUTYCYCLE_MAX) *pDutyCycleStepper
            = MCPWM_DUTYCYCLE_MAX;

    /* Save configuration in the structure */
    pStepperData->dutyCycleStepper = *pDutyCycleStepper;

    /* Must be the inverse of the CHANNEL 1 */
    dutyValCh1 = MCPWM_PRIMARY_PERIOD - *pDutyCycleStepper;

    PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL1,
            dutyValCh1);
    PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL2,
            *pDutyCycleStepper);
}

int16_t getStepperPower(STEPPER_DATA *pStepperData){

    return pStepperData->dutyCycleStepper;
}


//----------------------------------------------------------------------------// getStepperStruct
STEPPER_DATA* getMyStepperStruct(void){

    /* Return the address of the structure */
    return &stepperData;
}
```