Robotic Spider Control System Documentation

Overview

This lab involves creating a software system to control a robotic spider using memory-mapped I/O (MMIO) on a DE10-Nano SoC board. The codebase includes classes to interface with hardware components, manage motor operations, and coordinate movements of the spider robot.

File Structure

- `Main.cpp` : The entry point of the application, initializing the spider and handling user commands.
- `Spider.cpp` : Defines the `Spider` class, orchestrating the movements of the robot by controlling its legs.
- `SpiderLeg.cpp` : Implements the `SpiderLeg` class, representing a single leg composed of multiple joints.
- `ServoMotor.cpp` : Contains the `ServoMotor` class, managing individual servo motors via MMIO.
- `MMap.cpp` and `MMap.h` : Define the `MMap` class for handling memory mapping of device registers.
- `hps_0.h` : Provides hardware-specific definitions required for MMIO.
- `Makefile` : Contains build instructions for compiling the project.

Classes and Functionality

`MMap` Class (MMap.h)

Manages memory-mapped I/O operations:

- **Methods:**
  - `map(addr_base, addr_span)` : Establishes a memory mapping to the specified physical address range.
  - `unmap()` : Releases the existing memory mapping.
  - `isMapped()` : Checks if the mapping has been established.
  - `getMotorStart(motorId)` : Computes the virtual address of a motor's first register.
  - `Motor_Reg32_Write(motorId, regOffset, value)` : Writes a 32-bit value to a motor's register.

- `Motor_Reg32_Read(motorId, regOffset)` : Reads a 32-bit value from a motor's register.

`ServoMotor` Class (ServoMotor.cpp)

Controls individual servo motors:

- **Attributes:**
  - `m_fAngle` : Current angle of the servo motor.
  - `m_speed` : Operating speed of the servo motor.
  - `m_nMotorID` : Motor ID corresponding to the hardware PWM index.
  - `_mmio` : Pointer to an `MMap` instance for MMIO operations.
- **Methods:**
  - `ServoMotor(mmio, motorId)` : Constructor initializing the motor and setting default PWM values.
  - `Move(fAngle)` : Moves the servo to a specified angle, updating the PWM duty cycle.
  - `SetSpeed(speed)` : Sets the operating speed by updating the PWM delay.
  - `IsReady()` : Checks if the servo has completed its last movement.
  - `Reset()` : Resets the servo to its default position.
  - `speedToDelay(s)` : Converts a speed value to an appropriate delay in clock cycles.

`SpiderLeg` Class (SpiderLeg.cpp)

Represents a leg with three joints (hip, knee, ankle):

- **Attributes:**
  - `m_szMotor` : Array of `ServoMotor` instances for each joint.
  - `m_reverse` : Indicates if angle interpretations should be reversed.
- **Methods:**
  - `SpiderLeg(mmio, Joint0_MotorID, Joint1_MotorID, Joint2_MotorID, reverse)` : Constructor initializing motors.
  - `MoveJoint(JointID, fAngle)` : Moves a specific joint to a given angle.
  - `IsReady()` : Checks if all joints have completed movements.
  - `Reset()` : Resets all joints to default positions.
  - `GetfAngle(JointID)` : Gets the current angle of a specific joint.

`Spider` Class (Spider.cpp)

Controls the entire spider robot:

- **Attributes:**
  - `m_szLeg` : Array of `SpiderLeg` instances for each leg.

    - `lastStep` , `lastDir` : Track the last movement step and direction.
    - `_mmio` : Pointer to an `MMap` instance.
- **Methods:**
    - `Spider()` : Constructor initializing legs and MMIO interface.
    - `Init()` : Initializes the spider's legs to default positions.
    - `Standup()` : Moves the spider to a standing position.
    - `MoveForward()` : Coordinates legs for forward movement.
    - `MoveTripod(TripodID, JointID, AngleF, AngleM, AngleB)` : Moves a set of legs simultaneously.
    - `IsReady()` : Checks if all legs have completed movements.
    - `WaitReady()` : Waits until all movements are complete.
    - `Reset()` : Resets the spider to its initial position.

Constants and Definitions

- **Servo Limits:**
    - `DEGREE_MIN` : -90 degrees
    - `DEGREE_MAX` : 90 degrees
    - `SPEED_MIN` : 0
    - `SPEED_MAX` : 100
- **PWM Settings:**
    - `FREQ` : Clock frequency (50 MHz)
    - `T_20MS` : PWM period for a 20 ms cycle
    - `PWM_MIN` : Corresponds to -90 degrees
    - `PWM_MAX` : Corresponds to 90 degrees
- **Registers Offsets:**
    - `PWM_PERIOD` : Offset for PWM period register
    - `PWM_DC` : Offset for duty cycle register
    - `PWM_DELAY` : Offset for delay register
    - `PWM_READY` : Offset to check if the servo is ready
    - `PWM_ABORT` : Offset to abort operations

Building and Running

Build Instructions

Use the provided `Makefile` to compile the project:

```
make
```

Execution

Run the compiled executable:

```
./spider
```

Follow the on-screen prompts to control the spider:

Commands:

- `f` : Move forward
- `s` : Stop the application

Lab Objectives

- **Understanding MMIO:** Learn how to interface with hardware registers in C++ using memory-mapped I/O.
- **Servo Motor Control:** Implement control logic for servo motors using PWM signals.
- **Robotics Programming:** Coordinate multiple servos to perform complex movements.
- **Modular Design:** Develop a modular codebase that can be extended for various robotic applications.

Notes

- **Hardware Configuration:** Ensure motor IDs and hardware connections match the definitions in the code.
- **Parameter Adjustments:** Modify constants like `DELAY_MIN` and `DELAY_MAX` based on your pre-lab calculations and hardware specifications.
- **Safety Precautions:** Always test movements carefully to prevent damage to the hardware.

Summary

This lab provides hands-on experience with low-level hardware interfacing and robotics control. By understanding and manipulating MMIO, PWM signals, and servo mechanics, you gain valuable insights into embedded systems programming and robotic kinematics.