Lab 4

Servo Motors

Michael Kwok

ECE 315 Lab H41
Department of Electrical and Computer Engineering
University of Alberta
12 April 2021

1 Abstract

The Zybo Z7 is a digital logic and embedded software development platform by Digilent, containing a Zybo 7000 System on a Chip (SoC) that has both a digital logic fabric (Xilinx 7-series FPGA) and a hard processor (ARM Cortex-A9). For this lab, we will be making use of UART and GPIO for servo control

In the first part of the lab, a new task was to be created with the purpose of listening to a button to trigger an emergency stop. In the 2nd part of the lab, we write code to add simple "programmability" to the servo, by specifying steps and delays. The final part is to experimentally determine the limits of the servo.

All modified code for this lab has been attached in the Appendix.

2 Design

A new task was written to handle the input for emergency stoppage. The task used `vTaskDelayUntil` to keep a consistent frequency of polling, and kept listening until the button specified is held down for 3 intervals of that frequency. When that is detected, the task prints out to UART, informing the user that the motor is going to get stopped. The task accomplishes the stop by having a global flag that tells the other tasks if it has been triggered or not. Function calls to disable the other running tasks to ensure that the system is fully stopped, `vTaskDelete`, also gets called after execution of it's current loop is complete.

The program is then extended to handle extra inputs from the users relating to creating a test sequence for the motor. This is done by adding a field into the struct sent to the motor task which contains the delay expected for this command. The receiving task is also modified to handle multiple items in the queue, instead of the assumed single item by using an if statement instead of the blocking wait.

3 Testing

The motor's abilities was tested. This was done by attaching tape, a guitar pick and a paperclip to it, in that order. It was tested with the following targets: $512 \rightarrow 3000ms \rightarrow 1024 \rightarrow 1000ms \rightarrow 3072 \rightarrow 250ms \rightarrow 4096 \rightarrow 8192 \rightarrow 16384$. The results for all of them was found to be the same.

To find the result, the parameters are modified until skipping or weird behaviour is observed with the same targets. For all 3 items, the same point was observed at 500 for speed, 45000 for acceleration and seemingly unlimited for deceleration. The speed limit is due to physical constraints, while the acceleration limits are likely due to the code in `stepper.c`, within the `Stepper_processMovement()` function, which does not handle large numbers properly.

A Code

```
/*
 * main.c
 *
 *  Created on: Mar 24, 2021
 *  Author: Shyama Gandhi
 */

#include <stdbool.h>

#include "sleep.h"
#include "stepper.h"
#include "xgpio.h"  //GPIO functions definitions
#include "xil_cache.h"
#include "xparameters.h"  //DEVICE ID, UART BASEADDRESS, GPIO BASE ADDRESS definitions
#include "xuartps.h"  //UART definitions header file

static void _Task_Uart(void *pvParameters);
static TaskHandle_t xUarttask;

static void _Task_Motor(void *pvParameters);
static TaskHandle_t xMotortask;

static void vEmergencyStop(void *pvParameters);
static TaskHandle_t xEmergencyStopTask;

int Initialize_UART();

/************************ Queue Function definitions ************************/
static QueueHandle_t xQueue_FIFO1 = NULL;  // queue between task1 and task2

/************************ Global Variables **********************************/

// GPIO Button Instance and DEVICE ID
XGpio BTNInst;
#define EMERGENCY_STOP_BUTTON_DEVICE_ID XPAR_PMOD_BUTTONS_DEVICE_ID

// GPIO RGB led Instance and DEVICE ID
XGpio Red_RGBInst;
#define RGB_LED_DEVICE_ID XPAR_PMOD_RGB_DEVICE_ID

// struct for motor parameters
typedef struct {
  long currentposition_in_steps;
  float rotational_speed;
  float rotational_acceleration;
  float rotational_deceleration;
  long targetposition_in_steps;
  int delay_ms;
} decision_parameters;

int parameters_flag = 0;

bool emergencyStopped = false;

//-----------------------------------------------------
// MAIN FUNCTION
```

```c
//-----------------------------------------------------
int main(void) {
  int status;
  //-----------------------------------------------------------
  // INITIALIZE THE PMOD GPIO PERIPHERAL FOR STEPPER MOTOR, STOP BUTTON AND RGB
  // LED(that will flash the red light when emergency stop button is pushed
  // three times).
  //-----------------------------------------------------------

  // Initialize the PMOD for motor signals (JC PMOD is being used)
  status = XGpio_Initialize(&PModMotorInst, PMOD_MOTOR_DEVICE_ID);
  if (status != XST_SUCCESS) {
    xil_printf("GPIO Initialization for PMOD unsuccessful.\r\n");
    return XST_FAILURE;
  }

  // button for emergency stop activation
  // Initialize the PMOD for getting the button value (btn0 is being used)
  status = XGpio_Initialize(&BTNInst, EMERGENCY_STOP_BUTTON_DEVICE_ID);
  if (status != XST_SUCCESS) {
    xil_printf("GPIO Initialization for BUTTONS unsuccessful.\r\n");
    return XST_FAILURE;
  }

  // RGB Led for flashing the red light when stop button is activated
  // Initialize the PMOD for flashing the RED light on RGB LEDz
  status = XGpio_Initialize(&Red_RGBInst, RGB_LED_DEVICE_ID);
  if (status != XST_SUCCESS) {
    xil_printf("GPIO Initialization for BUTTONS unsuccessful.\r\n");
    return XST_FAILURE;
  }

  // Initialize the UART
  status = Initialize_UART();
  if (status != XST_SUCCESS) {
    xil_printf("UART Initialization failed\n");
  }

  // Set all buttons direction to inputs
  XGpio_SetDataDirection(&BTNInst, 1, 0xFF);
  // Set the RGB LED direction to output
  XGpio_SetDataDirection(&Red_RGBInst, 1, 0x00);

  xil_printf(
      "\nStepper motor Initialization Complete! Operational parameters can be "
      "changed below:\n\n");

  xTaskCreate(_Task_Uart, (const char *)"Uart Task",
              configMINIMAL_STACK_SIZE * 10, NULL, tskIDLE_PRIORITY + 1,
              &xUarttask);

  xTaskCreate(_Task_Motor, (const char *)"Motor Task",
              configMINIMAL_STACK_SIZE * 10, NULL, tskIDLE_PRIORITY + 2,
              &xMotortask);

  xTaskCreate(vEmergencyStop, (const char *)"Emergency Stop",
              configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 3,
              &xEmergencyStopTask);
```

```
115
116     // the queue size if set to 25 right now, you can change this size later on
117     // based on your requirements.
118
119     xQueue_FIFO1 =
120         xQueueCreate(20, sizeof(decision_parameters));  // connects task1 -> task2
121
122     configASSERT(xQueue_FIFO1);
123
124     vTaskStartScheduler();
125
126     while (1)
127       ;
128
129     return 0;
130   }
131
132   static void _Task_Uart(void *pvParameters) {
133     int message_flag = 0;
134     int commandsToSend = 0;
135     // this flag when a negative step value for target position is entered.
136     int direction_ccw_flag = 0;
137
138     decision_parameters params[20];
139
140     params[commandsToSend].currentposition_in_steps = 0;
141     params[commandsToSend].rotational_speed = 500;
142     params[commandsToSend].rotational_acceleration = 150;
143     params[commandsToSend].rotational_deceleration = 150;
144     params[commandsToSend].targetposition_in_steps =
145         NO_OF_STEPS_PER_REVOLUTION_FULL_DRIVE;
146     params[commandsToSend].delay_ms = 0;
147
148     while (!emergencyStopped) {
149       if (message_flag == 0) {
150         if (parameters_flag == 0) {
151           xil_printf("Current position of the motor = %d steps\n",
152                      params[commandsToSend].currentposition_in_steps);
153           xil_printf(
154               "Press <ENTER> to keep this value, or type a new starting position "
155               "and then <ENTER>\n");
156         } else if (parameters_flag == 1) {
157           printf("Current maximum speed of the motor = %0.1f steps/sec\n",
158                  params[commandsToSend].rotational_speed);
159           xil_printf(
160               "Press <ENTER> to keep this value, or type a new maximum speed "
161               "number and then <ENTER>\n");
162         } else if (parameters_flag == 2) {
163           printf(
164               "Current maximum acceleration of the motor = %0.1f steps/sec/sec\n",
165               params[commandsToSend].rotational_acceleration);
166           xil_printf(
167               "Press <ENTER> to keep this value, or type a new maximum "
168               "acceleration and then <ENTER>\n");
169         } else if (parameters_flag == 3) {
170           printf(
171               "Current maximum deceleration of the motor = %0.1f steps/sec/sec\n",
172               params[commandsToSend].rotational_deceleration);
```

```
          xil_printf(
              "Press <ENTER> to keep this value, or type a new maximum "
              "deceleration and then <ENTER>\n");
        } else if (parameters_flag == 4) {
          xil_printf("Destination position of the motor = %d steps\n",
                     params[commandsToSend].targetposition_in_steps);
          xil_printf(
              "Press <ENTER> to enter new value, or type a new destination "
              "position and then <ENTER>\n");
        } else if (parameters_flag == 5) {
          xil_printf("Delay at this position = %d ms\n", 0);
          xil_printf(
              "Press <ENTER> to keep this value, or type a new delay and then "
              "<ENTER>\n");
        } else if (parameters_flag == 6) {
          xil_printf(
              "Press <ENTER> to send all destinations, or type a new destination "
              "and then <ENTER>\n");
        }
      }

      char str_value_motor_value[] = "";
      char read_UART_character[100];  // an approximate size is being taken into
                                      // consideration. You will use a larger size
                                      // if you require.
      int invalid_input_flag = 0;

      int keep_default_value_flag = 0;
      int idx = 0;
      while (1) {
        if (XUartPs_IsReceiveData(XPAR_XUARTPS_0_BASEADDR)) {
          read_UART_character[idx] =
              XUartPs_ReadReg(XPAR_XUARTPS_0_BASEADDR, XUARTPS_FIFO_OFFSET);
          idx++;
          if (read_UART_character[idx - 1] == 0x0D) {
            break;
          }
        }
      }

      if (idx == 1) {
        if (read_UART_character[idx - 1] == 0x0D) {
          keep_default_value_flag = 1;
          invalid_input_flag = 0;
        }
      } else {
        if (parameters_flag < 4) {
          for (int i = 0; i < idx - 1; i++) {
            if (!(read_UART_character[i] >= '0' &&
                  read_UART_character[i] <= '9')) {
              invalid_input_flag = 1;
              break;
            } else {
              strncat(str_value_motor_value, &read_UART_character[i], 1);
              invalid_input_flag = 0;
            }
          }
        } else if (parameters_flag == 4 || parameters_flag == 5 ||
```

```
                parameters_flag == 6) {
      int iterate_index = 0;
      if (read_UART_character[0] == '-') {
        direction_ccw_flag = 1;
        iterate_index = 1;
      } else
        iterate_index = 0;

      for (int i = iterate_index; i < idx - 1; i++) {
        if (!(read_UART_character[i] >= '0' &&
              read_UART_character[i] <= '9')) {
          invalid_input_flag = 1;
          break;
        } else {
          strncat(str_value_motor_value, &read_UART_character[i], 1);
          invalid_input_flag = 0;
        }
      }
    }
  }

  if (invalid_input_flag == 1) {
    message_flag = 1;
    xil_printf(
        "There was an invalid input from user except the valid inputs "
        "between 0-9\n");
    xil_printf("Please input the value of this parameter again!\n");
  } else {
    message_flag = 0;
    parameters_flag += 1;
    if (parameters_flag == 1) {
      if (keep_default_value_flag == 1) {
        xil_printf(
            "User chooses to keep the default value of current position = %d "
            "steps\n\n",
            params[commandsToSend].currentposition_in_steps);
      } else {
        params[commandsToSend].currentposition_in_steps =
            atoi(str_value_motor_value);
        xil_printf("User entered the new current position = %d steps\n\n",
                   params[commandsToSend].currentposition_in_steps);
      }
    } else if (parameters_flag == 2) {
      if (keep_default_value_flag == 1) {
        printf(
            "User chooses to keep the default value of rotational speed = "
            "%0.1f steps/sec\n\n",
            params[commandsToSend].rotational_speed);
      } else {
        params[commandsToSend].rotational_speed = atoi(str_value_motor_value);
        printf("User entered the new rotational speed = %0.1f steps/sec\n\n",
               params[commandsToSend].rotational_speed);
      }
    } else if (parameters_flag == 3) {
      if (keep_default_value_flag == 1) {
        printf(
            "User chooses to keep the default value of rotational "
            "acceleration = %0.1f steps/sec/sec\n\n",
```

```
289                 params[commandsToSend].rotational_acceleration);
290           } else {
291             params[commandsToSend].rotational_acceleration =
292                 atoi(str_value_motor_value);
293             printf(
294                 "User entered the new rotational acceleration = %0.1f "
295                 "steps/sec/sec\n\n",
296                 params[commandsToSend].rotational_acceleration);
297           }
298         } else if (parameters_flag == 4) {
299           if (keep_default_value_flag == 1) {
300             printf(
301                 "User chooses to keep the default value of rotational "
302                 "deceleration = %0.1f steps/sec/sec\n\n",
303                 params[commandsToSend].rotational_deceleration);
304           } else {
305             params[commandsToSend].rotational_deceleration =
306                 atoi(str_value_motor_value);
307             printf(
308                 "User entered the new rotational deceleration = %0.1f "
309                 "steps/sec/sec\n\n",
310                 params[commandsToSend].rotational_deceleration);
311           }
312         } else if (parameters_flag == 5) {
313           if (keep_default_value_flag == 1) {
314             xil_printf(
315                 "User chooses to keep the default value of destination position "
316                 "= %d\n\n",
317                 params[commandsToSend].targetposition_in_steps);
318           } else {
319             params[commandsToSend].targetposition_in_steps =
320                 atoi(str_value_motor_value);
321             if (direction_ccw_flag == 1) {
322               params[commandsToSend].targetposition_in_steps =
323                   -params[commandsToSend].targetposition_in_steps;
324               direction_ccw_flag = 0;
325             }
326             xil_printf("User entered the new destination position = %d steps\n\n",
327                        params[commandsToSend].targetposition_in_steps);
328           }
329         } else if (parameters_flag == 6) {
330           if (keep_default_value_flag == 1) {
331             xil_printf("User chooses to keep the default delay of = 0 ms\n\n");
332           } else {
333             params[commandsToSend].delay_ms = atoi(str_value_motor_value);
334             xil_printf("User entered the new delay of = %d ms\n\n",
335                        params[commandsToSend].delay_ms);
336           }
337         } else if (parameters_flag == 7) {
338           if (keep_default_value_flag == 1) {
339             xil_printf(
340                 "\n****************************** MENU "
341                 "******************************\n");
342             xil_printf(
343                 "1. Press m<ENTER> to change the motor parameters again.\n");
344             xil_printf("2. Press g<ENTER> to start the movement of the motor.\n");
345
346             char command_1_or_2_values[100];
```

```
            int index = 0;
            char command;
            while (1) {
              if (XUartPs_IsReceiveData(XPAR_XUARTPS_0_BASEADDR)) {
                command_1_or_2_values[index] =
                    XUartPs_ReadReg(XPAR_XUARTPS_0_BASEADDR, XUARTPS_FIFO_OFFSET);
                index++;
                if (command_1_or_2_values[index - 1] == 0x0D) {
                  if ((index > 2) | (index == 1)) {
                    index = 0;
                  } else if (index == 2) {
                    command = command_1_or_2_values[index - 2];
                    if ((command == 'm') | (command == 'g')) {
                      break;
                    } else {
                      index = 0;
                    }
                  }
                }
              }
            }

            if (command == 'm') {
              parameters_flag = 0;
            } else if (command == 'g') {
              commandsToSend++;
              for (int i = 0; i < commandsToSend; i++) {
                xQueueSendToBack(xQueue_FIFO1, &params[i], 0UL);
              }
              commandsToSend = 0;
              parameters_flag = 0;
              taskYIELD();
              Stepper_setCurrentPositionInSteps(0);
            }
          } else {
            commandsToSend++;

            params[commandsToSend].currentposition_in_steps = 0;
            params[commandsToSend].rotational_speed = 500;
            params[commandsToSend].rotational_acceleration = 150;
            params[commandsToSend].rotational_deceleration = 150;

            params[commandsToSend].targetposition_in_steps =
                atoi(str_value_motor_value);
            params[commandsToSend].delay_ms = 0;
            if (direction_ccw_flag == 1) {
              params[commandsToSend].targetposition_in_steps =
                  -params[commandsToSend].targetposition_in_steps;
              direction_ccw_flag = 0;
            }
            xil_printf("User entered the new destination position = %d steps\n\n",
                       params[commandsToSend].targetposition_in_steps);

            parameters_flag -= 2;
          }
        }
      }
      vTaskDelay(1);
```

```
405     }
406     vTaskDelete(NULL);
407   }
408
409   /*-----------------------------------------------------------*/
410   static void _Task_Motor(void *pvParameters) {
411     decision_parameters read_motor_parameters_from_queue;
412
413     while (!emergencyStopped) {
414       if (xQueueReceive(xQueue_FIFO1, &read_motor_parameters_from_queue, 0UL)) {
415         Stepper_PMOD_pins_to_output();
416         Stepper_Initialize();
417
418         xil_printf("\nStarting the Motor Rotation...\n");
419
420         Stepper_setSpeedInStepsPerSecond(
421             read_motor_parameters_from_queue.rotational_speed);
422         Stepper_setAccelerationInStepsPerSecondPerSecond(
423             read_motor_parameters_from_queue.rotational_acceleration);
424         Stepper_setDecelerationInStepsPerSecondPerSecond(
425             read_motor_parameters_from_queue.rotational_deceleration);
426         Stepper_setCurrentPositionInSteps(
427             read_motor_parameters_from_queue.currentposition_in_steps);
428         Stepper_SetupMoveInSteps(
429             read_motor_parameters_from_queue.targetposition_in_steps);
430
431         while (!Stepper_motionComplete()) {
432           Stepper_processMovement();
433         }
434         if (read_motor_parameters_from_queue.delay_ms > 0) {
435           vTaskDelay(pdMS_TO_TICKS(read_motor_parameters_from_queue.delay_ms));
436         }
437       }
438       vTaskDelay(1);
439     }
440
441     Stepper_disableMotor();
442     vTaskDelete(NULL);
443   }
444
445   static void vEmergencyStop(void *pvParameters) {
446     TickType_t xFrequency = pdMS_TO_TICKS(10);  // 100 Hz -> 10 ms
447     TickType_t xLastWakeTime = xTaskGetTickCount();
448
449     BaseType_t stopCounter = 0;
450
451     while (!emergencyStopped) {
452       vTaskDelayUntil(&xLastWakeTime, xFrequency);
453       u32 buttonState = XGpio_DiscreteRead(&BTNInst, 1);
454       if (buttonState != 0) {
455         stopCounter++;
456       } else {
457         stopCounter = 0;
458       }
459
460       if (stopCounter >= 3) {
461         emergencyStopped = true;
462         xil_printf("Stopping motor");
```

```
463         Stepper_SetupStop();
464         xLastWakeTime = xTaskGetTickCount();
465         xFrequency = pdMS_TO_TICKS(50);  // 20 Hz -> 50 ms
466       }
467     }
468
469     while (emergencyStopped) {
470       XGpio_DiscreteWrite(&Red_RGBInst, 1, 1);
471       vTaskDelayUntil(&xLastWakeTime, xFrequency);
472       XGpio_DiscreteWrite(&Red_RGBInst, 1, 0);
473       vTaskDelayUntil(&xLastWakeTime, xFrequency);
474     }
475   }
```