Table 1: Initial calculations of motor speeds

|  |  |  |
| --- | --- | --- |
|  | Period | Time number in code |
| Carrier Period | 20ms | 2560 |
| Min (5%) | 1ms | 128 |
| Max (10%) | 2ms | 256 |

Table 2: final calculations of motor speeds

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Constant Name | Value in code (dec) | Value in hex |
| Period | PERIOD | 2560 | A00 |
| PW for fast CCW | FAST\_CCW | 256 | 100 |
| PW for slow CCW | SLOW\_CCW | 198 | C6 |
| PW for stop | STOP | 192 | C0 |
| PW for slow CW | SLOW\_CW | 186 | BA |
| PW for fast CCW | FAST\_CW | 128 | 80 |

Code:

////////////////

//Willard Wider

//6/14/18

//ELEC3800

//Lab10

//PWM

////////////////

//pin 1.1 is the input to change the frequency

/\* DriverLib Includes \*/

**#include** <ti/devices/msp432p4xx/driverlib/driverlib.h>

/\* Standard Includes \*/

**#include** <stdint.h>

**#include** <stdbool.h>

//pre-defined timer values for duty cycle

**#define** FAST\_CCW 256//10% duty cycle -> 2ms

**#define** SLOW\_CCW 198//

**#define** STOP 192//7.5% duty cycle -> 1.5ms

**#define** SLOW\_CW 186//

**#define** FAST\_CW 128//5% duty cycle -> 1ms

**#define** PERIOD 2560//period of "carrier wave"

//2560 = 20ms

/\* Timer\_A PWM Configuration Parameter \*/

Timer\_A\_PWMConfig pwmConfig =

{

TIMER\_A\_CLOCKSOURCE\_SMCLK,

TIMER\_A\_CLOCKSOURCE\_DIVIDER\_1,

PERIOD,

TIMER\_A\_CAPTURECOMPARE\_REGISTER\_1,

TIMER\_A\_OUTPUTMODE\_RESET\_SET,

FAST\_CW//enabled a 5% duty cycle ->1ms

};

**int** **main**(**void**)

{

/\* Halting the watchdog \*/

MAP\_WDT\_A\_holdTimer();

/\* Setting MCLK to REFO at 128Khz for LF mode Setting SMCLK to 64Khz \*/

MAP\_CS\_setReferenceOscillatorFrequency(CS\_REFO\_128KHZ);

MAP\_CS\_initClockSignal(CS\_MCLK, CS\_REFOCLK\_SELECT, CS\_CLOCK\_DIVIDER\_1);

MAP\_CS\_initClockSignal(CS\_SMCLK, CS\_REFOCLK\_SELECT, CS\_CLOCK\_DIVIDER\_1);

MAP\_PCM\_setPowerState(PCM\_AM\_LF\_VCORE0);

/\* Configuring GPIO2.4 as peripheral output for PWM and P6.7 for button interrupt \*/

MAP\_GPIO\_setAsPeripheralModuleFunctionOutputPin(GPIO\_PORT\_P2, GPIO\_PIN4, GPIO\_PRIMARY\_MODULE\_FUNCTION);

MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);

MAP\_GPIO\_clearInterruptFlag(GPIO\_PORT\_P1, GPIO\_PIN1);

MAP\_GPIO\_enableInterrupt(GPIO\_PORT\_P1, GPIO\_PIN1);

/\* Configuring Timer\_A \*/

MAP\_Timer\_A\_generatePWM(TIMER\_A0\_BASE, &pwmConfig);

/\* Enabling interrupts and starting the watchdog timer \*/

MAP\_Interrupt\_enableInterrupt(INT\_PORT1);

MAP\_Interrupt\_enableMaster();

**while** (1)

{

}

}

/\* Port1 ISR - This ISR will progressively step up the duty cycle of the PWM

\* on a button press

\*/

**void** **PORT1\_IRQHandler**(**void**)

{

uint32\_t status = MAP\_GPIO\_getEnabledInterruptStatus(GPIO\_PORT\_P1);

MAP\_GPIO\_clearInterruptFlag(GPIO\_PORT\_P1, status);

**if** (status & GPIO\_PIN1)

{

//on button press

//switch statement to specify which one to move to

//rather than having a motor\_state variable, use the direct pwm frequency itself

**switch**(pwmConfig.dutyCycle)

{

**case** FAST\_CCW:

pwmConfig.dutyCycle = SLOW\_CCW;

**break**;

**case** SLOW\_CCW:

pwmConfig.dutyCycle = STOP;

**break**;

**case** STOP:

pwmConfig.dutyCycle = SLOW\_CW;

**break**;

**case** SLOW\_CW:

pwmConfig.dutyCycle = FAST\_CW;

**break**;

**case** FAST\_CW:

pwmConfig.dutyCycle = FAST\_CCW;

**break**;

}

MAP\_Timer\_A\_generatePWM(TIMER\_A0\_BASE, &pwmConfig);

}

}