**Date Submitted: 10/18/2018**

**Assignment Youtube Playlist:** **https://www.youtube.com/playlist?list=PL4oTyvRrubXc8Wz0KqtDVSOubTPp4v2b2**

# Task 00: Execute the supplied code, no submission required.

**Youtube Link:** **https://youtu.be/mBeeZguVI2s**

**Original Code (added comments):**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/pwm.h"

**#include** "driverlib/pin\_map.h"

**#include** "inc/hw\_gpio.h"

**#include** "driverlib/rom.h"

//Include all needed files for peripherals and system operations.

**#define** PWM\_FREQUENCY 55 //PWM frequency of 55 Hz, which is a period of 18.2ms.

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint8\_t ui8Adjust;

ui8Adjust = 83;

//Create our variables used by our PWM to control the servo. 83 is our center position to create an approximate

//1.5ms delay, the center position for our servo.

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

//Set system clock 40 MHz, PWM clock divided by 64 for 625 kHz.

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

//Enable clock to PWM1, GPIO ports D and F.

ROM\_GPIOPinTypePWM(GPIO\_PORTD\_BASE, GPIO\_PIN\_0);

ROM\_GPIOPinConfigure(GPIO\_PD0\_M1PWM0);

//Enable PWM to output to port D pin 0.

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = GPIO\_LOCK\_KEY;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_CR) |= 0x01;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = 0;

ROM\_GPIODirModeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_DIR\_MODE\_IN);

ROM\_GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

//Unlock PF0 for button use, configure PF0 and PF4 as inputs, and configure pull-up resistors for these pins.

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM1\_BASE, PWM\_GEN\_0, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM1\_BASE, PWM\_GEN\_0, ui32Load);

//Calculate our PWM clock then divide by PWM frequency to load into timer for PWM generation.

//PWM1 configured as down-counter, calculated load count loaded into PWM.

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_0\_BIT, true);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_0);

//Pull PWM load value, divide by 1000 to give us our minimum resolution (constrained by variable size).

//With a period of 18.2ms, each step will be 18.2us.

//Configure PWM1 generator then enable it.

**while**(1)

{

//If switch 1 is pressed, decrement our duty cycle to adjust servo.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_4)==0x00)

{

ui8Adjust--;

**if** (ui8Adjust < 56)

{

ui8Adjust = 56;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

}

//If switch 2 is pressed, increment our duty cycle to adjust servo.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_0)==0x00)

{

ui8Adjust++;

**if** (ui8Adjust > 111)

{

ui8Adjust = 111;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

}

ROM\_SysCtlDelay(100000); //Delay between actions, causes servo adjustment speed to be slowed by microcontroller delay.

}

}

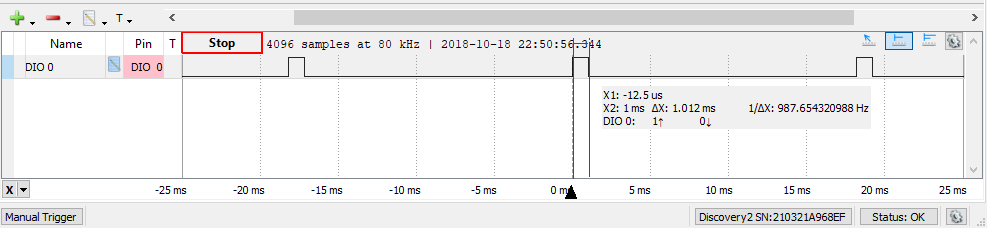


Figure - Minimum duty cycle

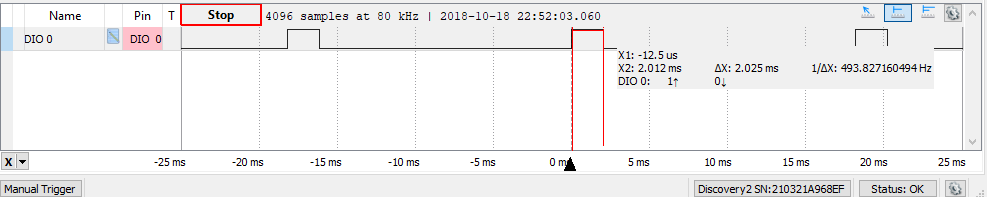


Figure - Maximum duty cycle

**-----------------------------------------------------------------------------------**

# Task 01: Change the PWM duty cycle to make the servo motor to do a loop of a complete sweep from 0 to 180 deg.

**Youtube Link:** **https://youtu.be/BY1iQvN1cQ0**

For Task 1 I modified the PWM duty cycle to allow a full 180° sweep. Many servo manufacturers do not officially support this full range of motion, but most of them can reach this range of motion, as long as care is taken to not damage the servos at the extreme ranges.

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/pwm.h"

**#include** "driverlib/pin\_map.h"

**#include** "inc/hw\_gpio.h"

**#include** "driverlib/rom.h"

//Include all needed files for peripherals and system operations.

**#define** PWM\_FREQUENCY 50 //PWM frequency of 50 Hz, which is a period of 20ms.

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint8\_t ui8Adjust;

ui8Adjust = 75;

//Create our variables used by our PWM to control the servo. 75 is our center position to create an approximate

//1.5ms delay, the center position for our servo.

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

//Set system clock 40 MHz, PWM clock divided by 64 for 625 kHz.

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

//Enable clock to PWM1, GPIO ports D and F.

ROM\_GPIOPinTypePWM(GPIO\_PORTD\_BASE, GPIO\_PIN\_0);

ROM\_GPIOPinConfigure(GPIO\_PD0\_M1PWM0);

//Enable PWM to output to port D pin 0.

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = GPIO\_LOCK\_KEY;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_CR) |= 0x01;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = 0;

ROM\_GPIODirModeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_DIR\_MODE\_IN);

ROM\_GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

//Unlock PF0 for button use, configure PF0 and PF4 as inputs, and configure pull-up resistors for these pins.

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM1\_BASE, PWM\_GEN\_0, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM1\_BASE, PWM\_GEN\_0, ui32Load);

//Calculate our PWM clock then divide by PWM frequency to load into timer for PWM generation.

//PWM1 configured as down-counter, calculated load count loaded into PWM.

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_0\_BIT, true);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_0);

//Pull PWM load value, divide by 1000 to give us our minimum resolution (constrained by variable size).

//With a period of 20ms, each step will be 20us.

//Configure PWM1 generator then enable it.

**while**(1)

{

//If switch 1 is pressed, decrement our duty cycle to adjust servo.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_4)==0x00)

{

ui8Adjust--;

**if** (ui8Adjust < 25)

{

ui8Adjust = 25;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

}

//If switch 2 is pressed, increment our duty cycle to adjust servo.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_0)==0x00)

{

ui8Adjust++;

**if** (ui8Adjust > 125)

{

ui8Adjust = 125;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_0, ui8Adjust \* ui32Load / 1000);

}

ROM\_SysCtlDelay(200000); //Delay between actions, causes servo speed to be slowed by microcontroller delay.

}

}

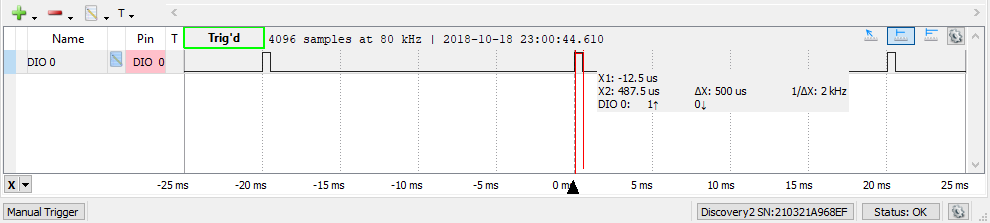


Figure - PWM minimum range

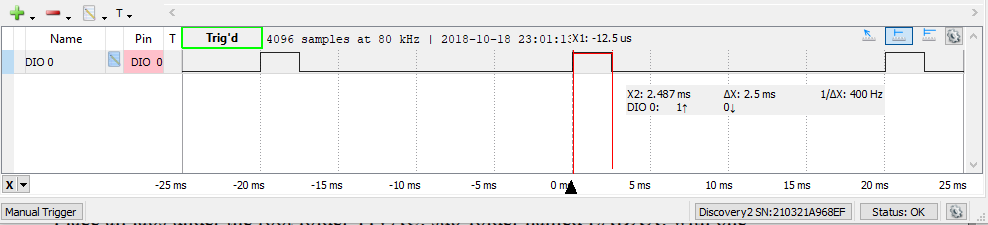


Figure - PWM maximum range

**-----------------------------------------------------------------------------------**

# Task 02: Change PWM duty cycle from 10% to 90% to control the brightness of the LED at PF1

**Youtube Link:** **https://youtu.be/Ofy8YnumGZE**

For Task 2 I modified our PWM signal to operate at 5kHz, to pulse the LED quickly enough for it to appear to be dimming rather than flickering to the human eye. The period is calculated and multiplied by our duty cycle variable, which adjusts between 10 and 90 by pressing the switches on our device.

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/pwm.h"

**#include** "driverlib/pin\_map.h"

**#include** "inc/hw\_gpio.h"

**#include** "driverlib/rom.h"

//Include all needed files for peripherals and system operations.

**#define** PWM\_FREQUENCY 5000 //PWM frequency of 5 kHz, which is a period of 20us.

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint8\_t ui8Adjust;

ui8Adjust = 50;

//Create our variables used by our PWM to control the LED. Begin at 50%.

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

//Set system clock 40 MHz, PWM clock divided by 64 for 625 kHz.

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

//Enable clock to PWM5, GPIO ports D and F.

ROM\_GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

//Enable PWM to output to port F pin 1.

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = GPIO\_LOCK\_KEY;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_CR) |= 0x01;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = 0;

ROM\_GPIODirModeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_DIR\_MODE\_IN);

ROM\_GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

//Unlock PF0 for button use, configure PF0 and PF4 as inputs, and configure pull-up resistors for these pins.

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

**PWMGenConfigure**(PWM1\_BASE, PWM\_GEN\_2, PWM\_GEN\_MODE\_DOWN);

**PWMGenPeriodSet**(PWM1\_BASE, PWM\_GEN\_2, ui32Load);

//Calculate our PWM clock then divide by PWM frequency to load into timer for PWM generation.

//PWM1 configured as down-counter, calculated load count loaded into PWM.

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui8Adjust \* ui32Load / 100);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_5\_BIT, true);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_2);

//Pull PWM load value, divide by 100 to give us our minimum resolution (constrained by variable size).

//With a period of 20us, each step will be 200ns.

//Configure PWM1 generator then enable it.

**while**(1)

{

//If switch 1 is pressed, decrement our duty cycle to adjust LED.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_4)==0x00)

{

ui8Adjust--;

**if** (ui8Adjust < 10)

{

ui8Adjust = 10;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui8Adjust \* ui32Load / 100);

}

//If switch 2 is pressed, increment our duty cycle to adjust LED.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_0)==0x00)

{

ui8Adjust++;

**if** (ui8Adjust > 90)

{

ui8Adjust = 90;

}

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui8Adjust \* ui32Load / 100);

}

ROM\_SysCtlDelay(200000); //Delay between actions, causes LED adjustment speed to be slowed by microcontroller delay.

}

}

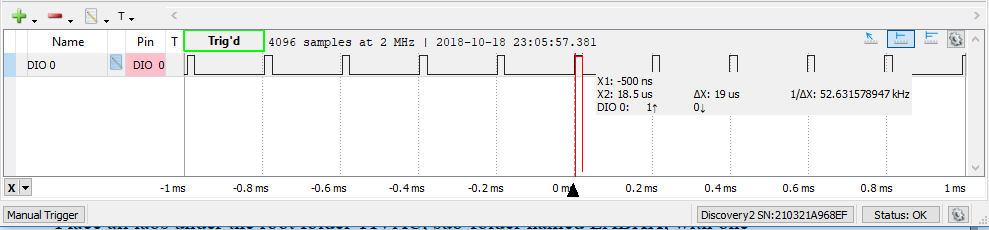


Figure - Red LED PWM minimum

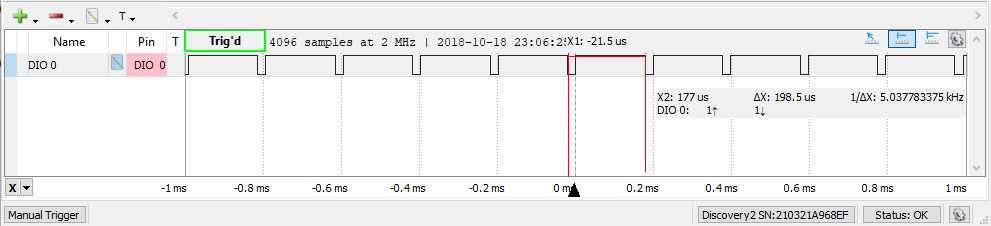


Figure - Red LED PWM maximum

**------------------------------------------------------------------------------------**

# Task 03: Change PWM duty cycle from 90% to 10% to control the brightness of the all three LED at PF1, PF2, and PF3 using three nested “for loops”.

**Youtube Link:** **https://youtu.be/DMz7rV1bvBE**

For Task 3 we enabled the PWM generators required for the green and blue LEDs as well as the red LED, using three nested FOR-loops to cycle through all possible combinations of RBG between 10% and 90% duty cycle, with 1% resolution. The nested FOR-loops create a bit of a sudden change when each LED moves quickly from 10% back to 90%, which could be smoothed out in later iterations of the code by transitioning slowly back up to 90% rather than moving in a single increment.

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/pwm.h"

**#include** "driverlib/pin\_map.h"

**#include** "inc/hw\_gpio.h"

**#include** "driverlib/rom.h"

//Include all needed files for peripherals and system operations.

**#define** PWM\_FREQUENCY 5000 //PWM frequency of 5 kHz, which is a period of 20us.

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

{

**volatile** uint32\_t ui32Load;

**volatile** uint32\_t ui32PWMClock;

**volatile** uint8\_t ui8Adjust;

**volatile** uint8\_t r;

**volatile** uint8\_t b;

**volatile** uint8\_t g;

ui8Adjust = 90;

//Create our variables used by our PWM to control the LED. Begin at 90%.

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPWMClockSet(SYSCTL\_PWMDIV\_64);

//Set system clock 40 MHz, PWM clock divided by 64 for 625 kHz.

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM1);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

//Enable clock to PWM5, GPIO ports D and F.

ROM\_GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

ROM\_GPIOPinConfigure(GPIO\_PF1\_M1PWM5);

ROM\_GPIOPinConfigure(GPIO\_PF2\_M1PWM6);

ROM\_GPIOPinConfigure(GPIO\_PF3\_M1PWM7);

//Enable PWM to output to port F pins 1,2,3.

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = GPIO\_LOCK\_KEY;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_CR) |= 0x01;

HWREG(GPIO\_PORTF\_BASE + GPIO\_O\_LOCK) = 0;

ROM\_GPIODirModeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_DIR\_MODE\_IN);

ROM\_GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4|GPIO\_PIN\_0, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

//Unlock PF0 for button use, configure PF0 and PF4 as inputs, and configure pull-up resistors for these pins.

ui32PWMClock = **SysCtlClockGet**() / 64;

ui32Load = (ui32PWMClock / PWM\_FREQUENCY) - 1;

ROM\_PWMGenConfigure(PWM1\_BASE, PWM\_GEN\_2, PWM\_GEN\_MODE\_DOWN);

ROM\_PWMGenConfigure(PWM1\_BASE, PWM\_GEN\_3, PWM\_GEN\_MODE\_DOWN);

ROM\_PWMGenPeriodSet(PWM1\_BASE, PWM\_GEN\_2, ui32Load);

ROM\_PWMGenPeriodSet(PWM1\_BASE, PWM\_GEN\_3, ui32Load);

//Calculate our PWM clock then divide by PWM frequency to load into timer for PWM generation.

//PWM1 configured as down-counter, calculated load count loaded into PWM.

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, ui8Adjust \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, ui8Adjust \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_7, ui8Adjust \* ui32Load / 100);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_5\_BIT, true);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_6\_BIT, true);

ROM\_PWMOutputState(PWM1\_BASE, PWM\_OUT\_7\_BIT, true);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_2);

ROM\_PWMGenEnable(PWM1\_BASE, PWM\_GEN\_3);

//Pull PWM load value, divide by 100 to give us our minimum resolution (constrained by variable size).

//With a period of 20us, each step will be 200ns.

//Configure PWM1 generators 2,3 then enable them.

**while**(1)

{

//If switch 1 is pressed, decrement our duty cycle to adjust LED.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_4)==0x00)

{

**for**(r=90; r>=10; r--)

{

**for**(b=90; b>=10; b--)

{

**for**(g=90; g>=10; g--)

{

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, r \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, b \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_7, g \* ui32Load / 100);

}

}

}

}

//If switch 2 is pressed, increment our duty cycle to adjust LED.

**if**(ROM\_GPIOPinRead(GPIO\_PORTF\_BASE,GPIO\_PIN\_0)==0x00)

{

**for**(r=10; r<=90; r++)

{

**for**(b=10; b<=90; b++)

{

**for**(g=10; g<=90; g++)

{

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_5, r \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, b \* ui32Load / 100);

ROM\_PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_7, g \* ui32Load / 100);

}

}

}

}

}

}

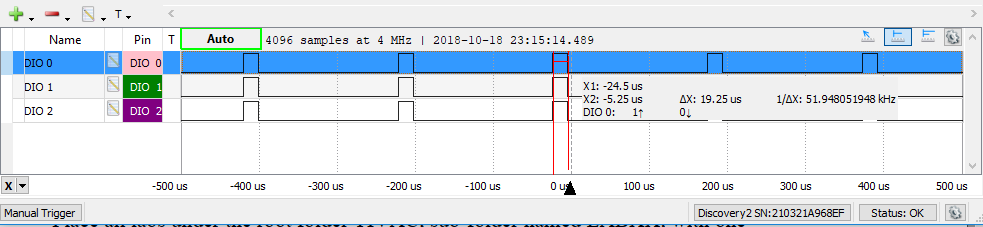


Figure - Minimum total duty cycle

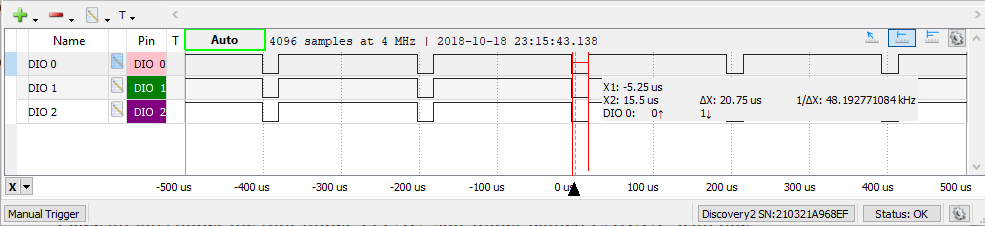


Figure - Maximum total duty cycle

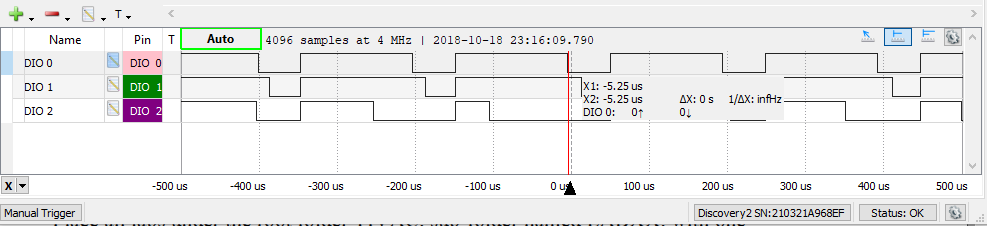


Figure - Transitionary duty cycle