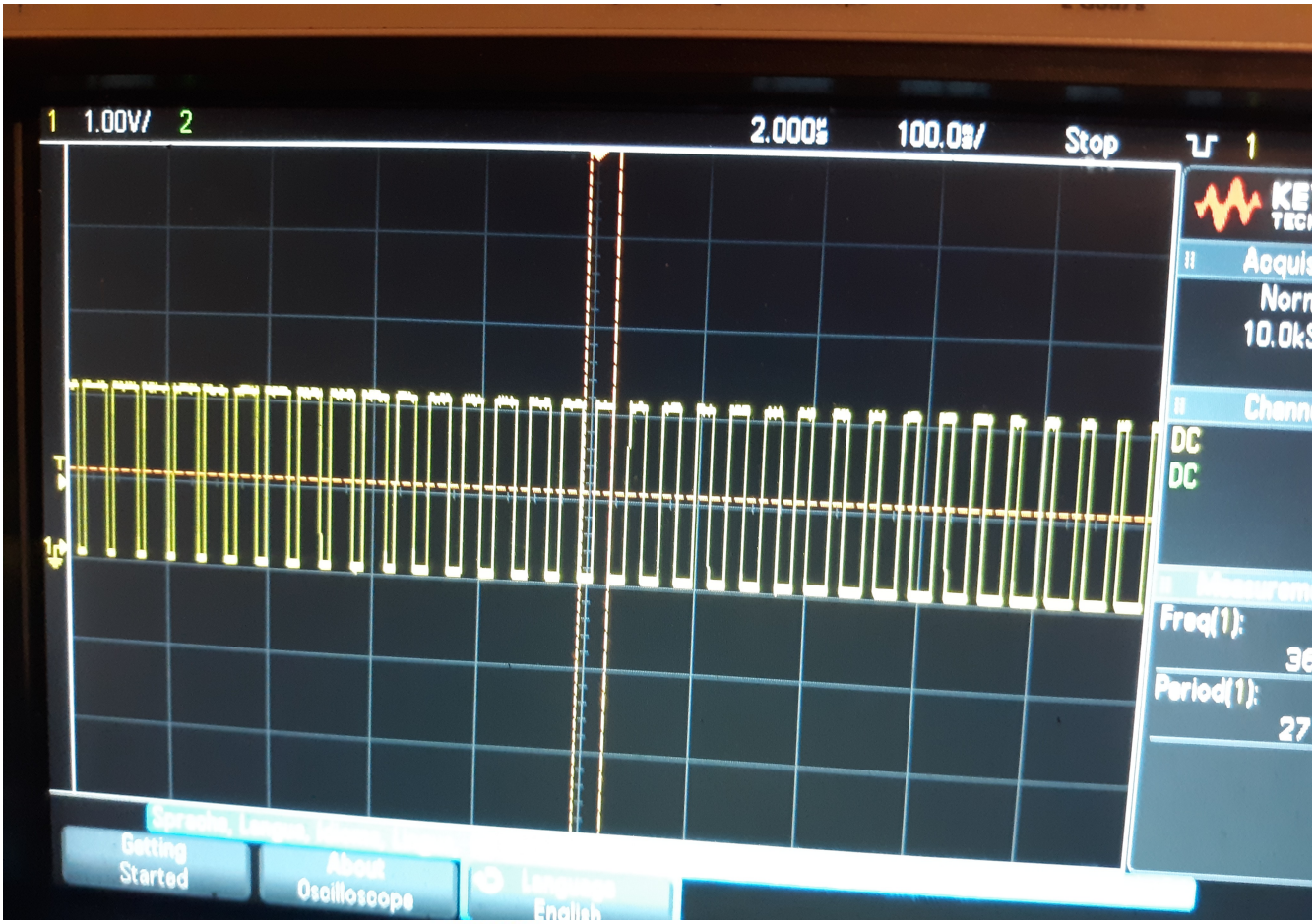
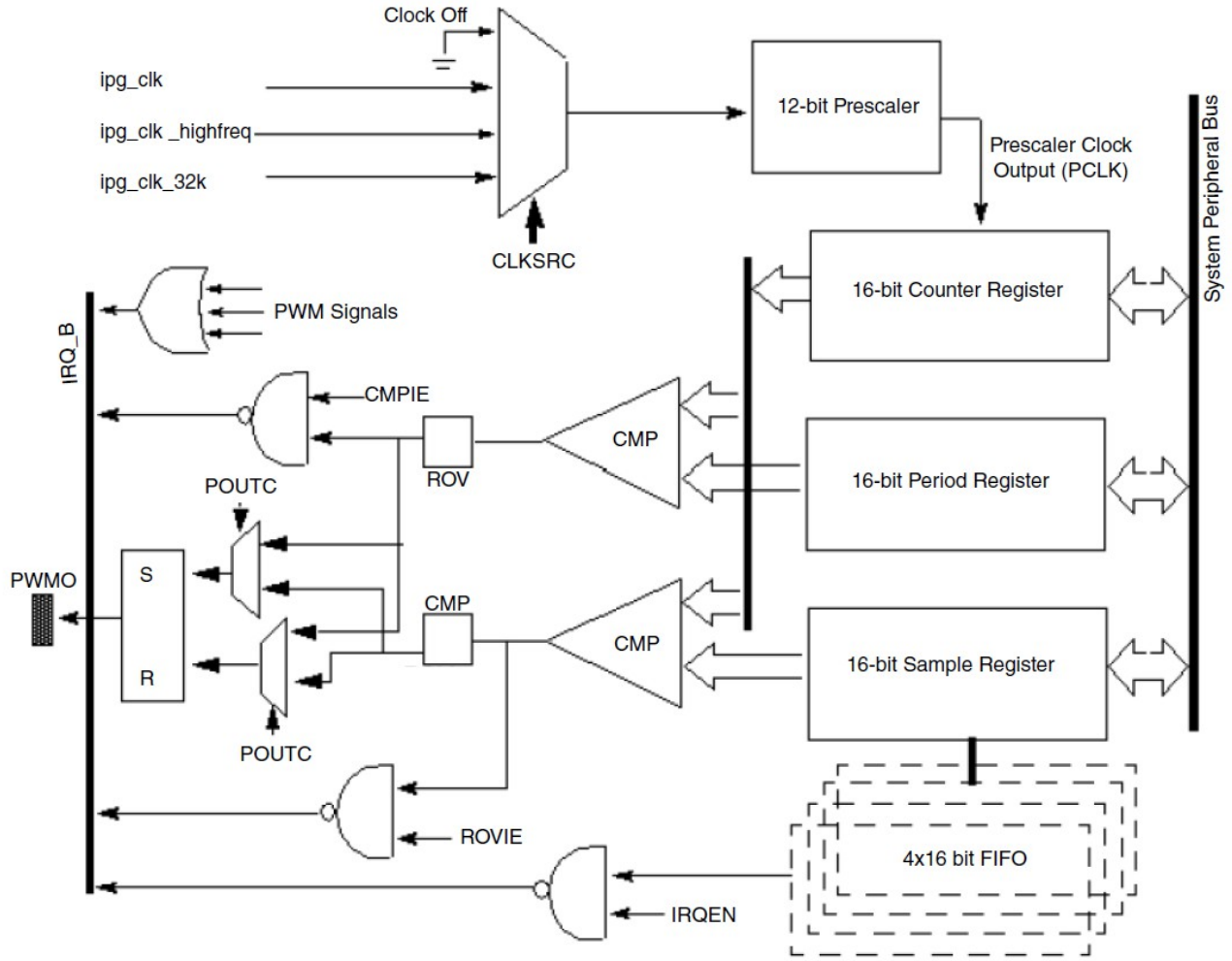

1 1.00V/ 2
2.000s
100.0s/
Stop
1
Getting Started
About Oscilloscope
Freq(1):
Period(1):


Clock Off
ipg_clk
ipg_clk _highfreq
ipg_clk_32k
CLKSRC
12-bit Prescaler
Prescaler Clock Output (PCLK)
System Peripheral Bus
16-bit Counter Register
16-bit Period Register
16-bit Sample Register
4x16 bit FIFO
PWM Signals
IRQ_B
CMPIE
ROV
CMP
POUTC
PWMO
S
R
CMP
CMP
POUTC
ROVIE
IRQEN

This is a Scalable PWM Modulator based on the infamous NXP PWM Peripheral present on an Toradex Colibri imx7 module, M4 core suitable for driving current through LED(s) or controlling brightness/contrast on a display and even perhaps motor control with some adjustments.

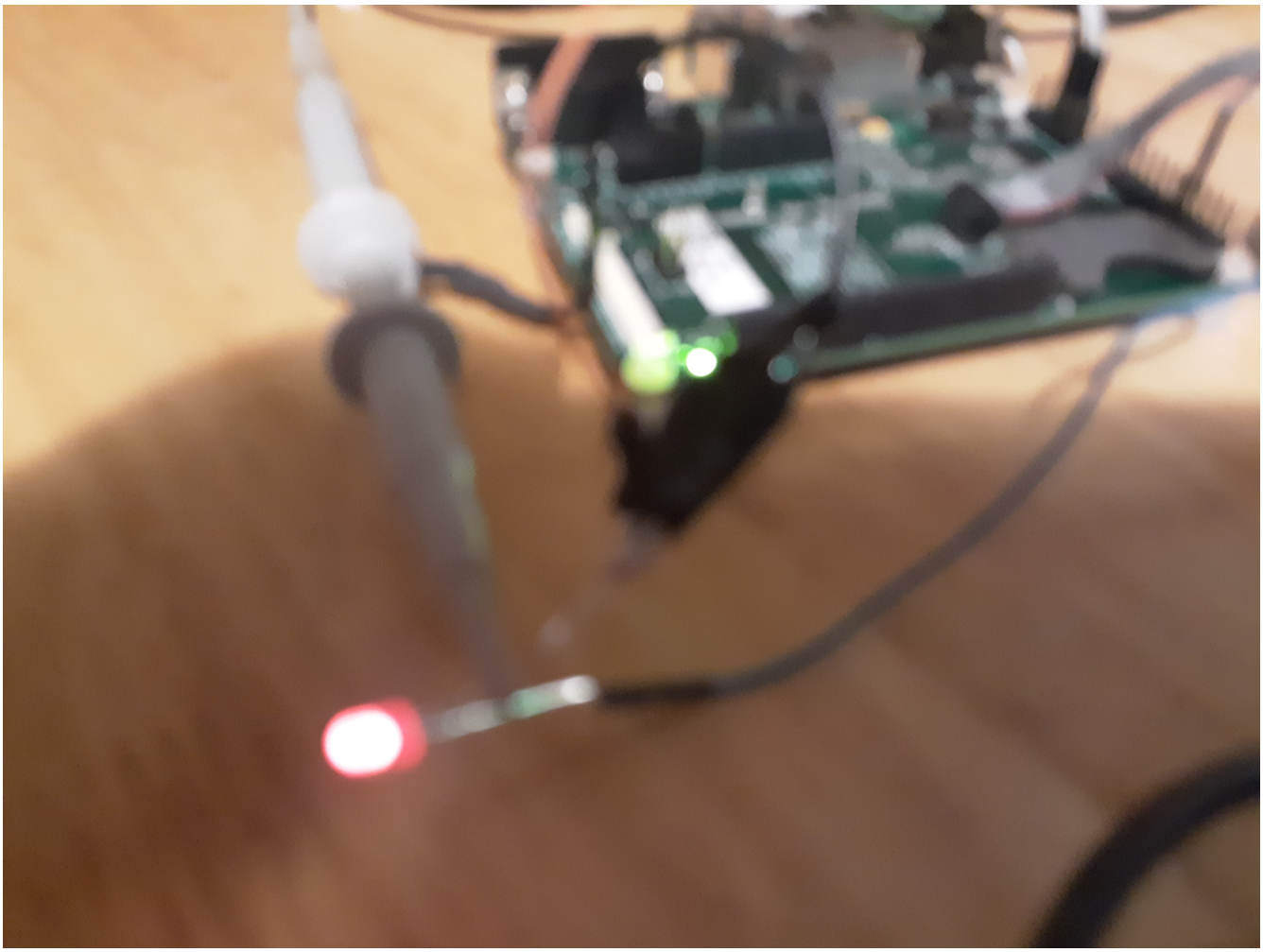
Make sure you get the pinning correct as I have changed the settings to allow PWM LED routing to the PWM_B Raspberry PI header at PIN Position 32. If your LED burns, assume you routed incorrectly...Not assuming any responsibility for any damage that may happen to your board! An improved, multi-channel controller/driver is on the way soon! Email me at : solaraeng@gmail.com if any issues or improvements you discovered...Enjoy!

Features :

1. Better Response/Resolution and less Peripheral jogging/dependencies
2. Much improved LED intensity at PEAK PWM duty-cycle...

Still To Do:

(Offer a 4-channel driver that is more generic in use)



```
//*
```

```
 * Copyright (c) 2015, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * ***************************************************************************
 * Project - mx7_colibri_m4_PWM_periph_demo (NXP PWM Peripheral LED Modulator)
 * Created by : Mario Ghecea
 * Solara Engineering (solaraeng@gmail.com)
 * 7/3/2019
 * Purpose - To facilitate a scalable and programmable PWM peripheral algorithm excluding
     FreeRTOS
 * utilizing any number of dividing steps (1-n) for smoothness and PWM resolution
 * Only one PWM Counter for the period and a samples FIFO for adequate phase
     synchronization
 * is used while keeping track of kPWM_FIFOEmptyFlag to enter the next phase.
 * This time the PWM interrupt is used as the feeder system to the samples FIFO which
 * results into a much smoother response and PWM precision.
 *
 * I use a PWM duty-cycle update delay inside the integrator. This results into a nice
 * accordion like modulation display which I find quite pleasant...
 * This could be used as a generic LED driver, contrast for a display and perhaps
 * motor control through expansion.
 *
 * TO DO - Create a multi-channel driver unless someone else beats me to it!
 *
 * If you reuse or distribute for your purpose please keep this header...
 ****************************************************************************
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include "board.h"
#include "gpio_pins.h"
#include "gpio_imx.h"
#include "debug_console_imx.h"
```

```
#include "pwm_imx.h"

/*! @brief PWM period value. PWMO (Hz) = PCLK(Hz) / (period +2) */
#define PWM_PERIOD_DIV                  16
            // Choose a larger divider for a faster accordion-like brightness display
#define PWM_PERIOD_VALUE        (16000/PWM_PERIOD_DIV)                          // 1
second period/PWM_PERIOD_DIV
#define PWM_STEPS_PER_PHASE             10
      // Increment PWM_STEPS_PER_PHASE for a higher resolution
#define PWM_STEP_WIDTH                  (PWM_PERIOD_VALUE/PWM_STEPS_PER_PHASE) //
Accordion step width
#define PWM_DELAY_DIV                2
      // Make this value greater for faster accordion fold...
#define PWM_DELAY_CNTR                  (PWM_STEPS_PER_PHASE/PWM_DELAY_DIV)    // This
determines how fast accordion folds/unfolds

/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Variables
 ******************************************************************************/
volatile uint32_t pwmDutycycle = 0U;
volatile bool pwmDutyUp = true;         /* Indicate PWM Duty cycle is increase or decrease
*/
volatile uint8_t stepCounter = PWM_DELAY_CNTR;

/* button relevent variables */
#ifdef BOARD_GPIO_KEY_CONFIG
static volatile uint8_t button_pressed_flag;
#endif

/*******************************************************************************
 * Code
 ******************************************************************************/
//Note - All integration/Modulation Magic happens here almost automatically!
//       By clearing kPWM_FIFOEmptyFlag it guarantees a smooth re-entrancy to this ISR
//       to automatically, integrate the PWM Duty-cycle. All outputs go straight out of
        peripheral.
void BOARD_PWM2_HANDLER(void)
{
      //static long counter = 10;
       /* Gets interrupt kPWM_FIFOEmptyFlag */
      if(PWM_GetStatusFlags(BOARD_PWM2_BASEADDR) & kPWM_FIFOEmptyFlag)
      {

            if (stepCounter == 0U)
            {
                  stepCounter = PWM_DELAY_CNTR;

                  if(pwmDutyUp)
                  {
                        /* Increase duty cycle until it reach limited value. */
                        if((pwmDutycycle += PWM_STEP_WIDTH) >= PWM_PERIOD_VALUE)
                        {
                              pwmDutycycle = PWM_PERIOD_VALUE;
                              pwmDutyUp = false;
```

```
                        }
                    }
                    else // pwmDutyDn
                    {
                        /* Decrease duty cycle until it reach limited value. */
                        if((pwmDutycycle -= PWM_STEP_WIDTH) <= 0U)
                        {
                            pwmDutycycle = 0U;
                            pwmDutyUp = true;
                        }
                    }
                }
                else
                    stepCounter --; // Do all the step counts at same modulation ratio
                /* Write duty cycle to PWM sample register.  */
                PWM_SetSampleValue(BOARD_PWM2_BASEADDR, pwmDutycycle);
                PWM_clearStatusFlags(BOARD_PWM2_BASEADDR, kPWM_FIFOEmptyFlag);
        }

}


/*!
* Function Name: main
*****************************************************************************/
int main(void)
{

    /* hardware initialize */
    hardware_init();
    PRINTF("\n\r====================== PWM Peripheral driver Example
=======================\n\r");

    PWM_GetDefaultConfig(&pwmConfig);

    /* Initialize PWM module */
    PWM_Init(BOARD_PWM2_BASEADDR, &pwmConfig);

    inter = PWM_GetEnabledInterrupts(BOARD_PWM2_BASEADDR);

    /* Enable FIFO empty interrupt */
    PWM_EnableInterrupts(BOARD_PWM2_BASEADDR, kPWM_FIFOEmptyInterruptEnable);

    inter = PWM_GetEnabledInterrupts(BOARD_PWM2_BASEADDR);

    /* Initial samples be written to the PWM Sample Register */
    PWM_SetSampleValue(BOARD_PWM2_BASEADDR, pwmDutycycle);

      /* Three initial samples be written to the PWM Sample Register */
      for(pwmDutycycle = 0u; pwmDutycycle < 3; pwmDutycycle++)
      {
         PWM_SetSampleValue(BOARD_PWM2_BASEADDR, pwmDutycycle);
      }

      /* Check and Clear interrupt status flags */
      if(PWM_GetStatusFlags(BOARD_PWM2_BASEADDR))
      {
            PWM_clearStatusFlags(BOARD_PWM2_BASEADDR, kPWM_FIFOEmptyFlag |
```

```
kPWM_RolloverFlag | kPWM_CompareFlag | kPWM_FIFOWriteErrorFlag);
      }

    /* Write the period to the PWM Period Register */
    PWM_SetPeriodValue(BOARD_PWM2_BASEADDR, PWM_PERIOD_VALUE);

    /* Set PWM Interrupt priority */
    NVIC_SetPriority(BOARD_PWM2_IRQ_NUM, 5);

    /* Call core API to enable the IRQ. */
    NVIC_EnableIRQ(BOARD_PWM2_IRQ_NUM);

    /* Start PWM Output */
    PWM_StartTimer(BOARD_PWM2_BASEADDR);

    while (true)
    {


    };
}
```

PINOUT FOR LED: PIN (32) on Raspberry Pi Connector on Toradex ASTER...