Buck Converter and Constant-Current Driver

A.G.Dean
ECE 460/560
Embedded System Architectures

Switch-Mode Power Conversion and the Buck Converter

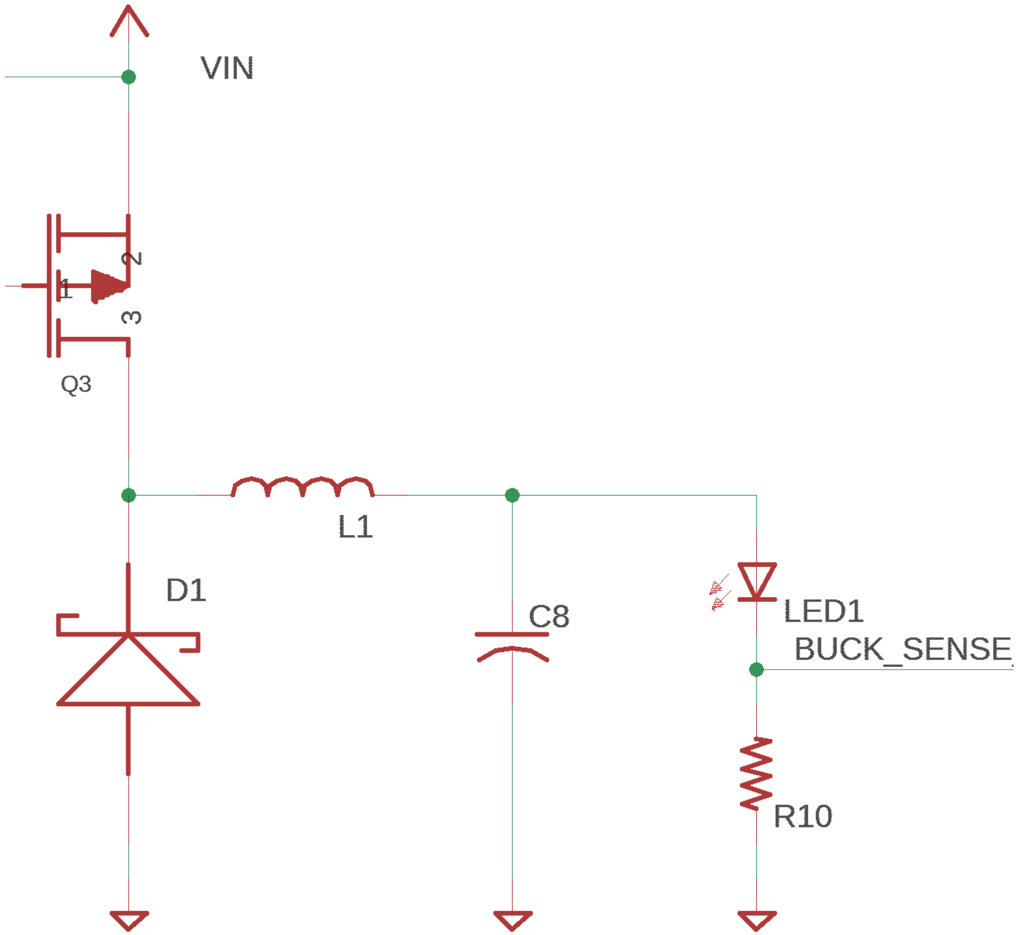
How to Lower a Voltage from V_{in} to V_{out}

- Linear regulator
  - Drops input voltage using “pass” transistor as variable resistor, converting extra power to heat.
  - If output doesn’t need as much power, reduce conductance of transistor (raise resistance), wasting more power.
- Switch-mode regulator
  - Drops voltage by connecting and disconnecting input voltage very frequently
  - If output doesn’t need as much power, reduce duty cycle (fraction of time) input is connected.
  - Minor additional power loss.
    - Switches (transistors, diodes) are either on (very low resistance) or off (open circuit)
    - Duty cycle typically controlled by pulse-width modulated (PWM) signal
  - If needed, filter output voltage with capacitor

Energy Storage and Filtering

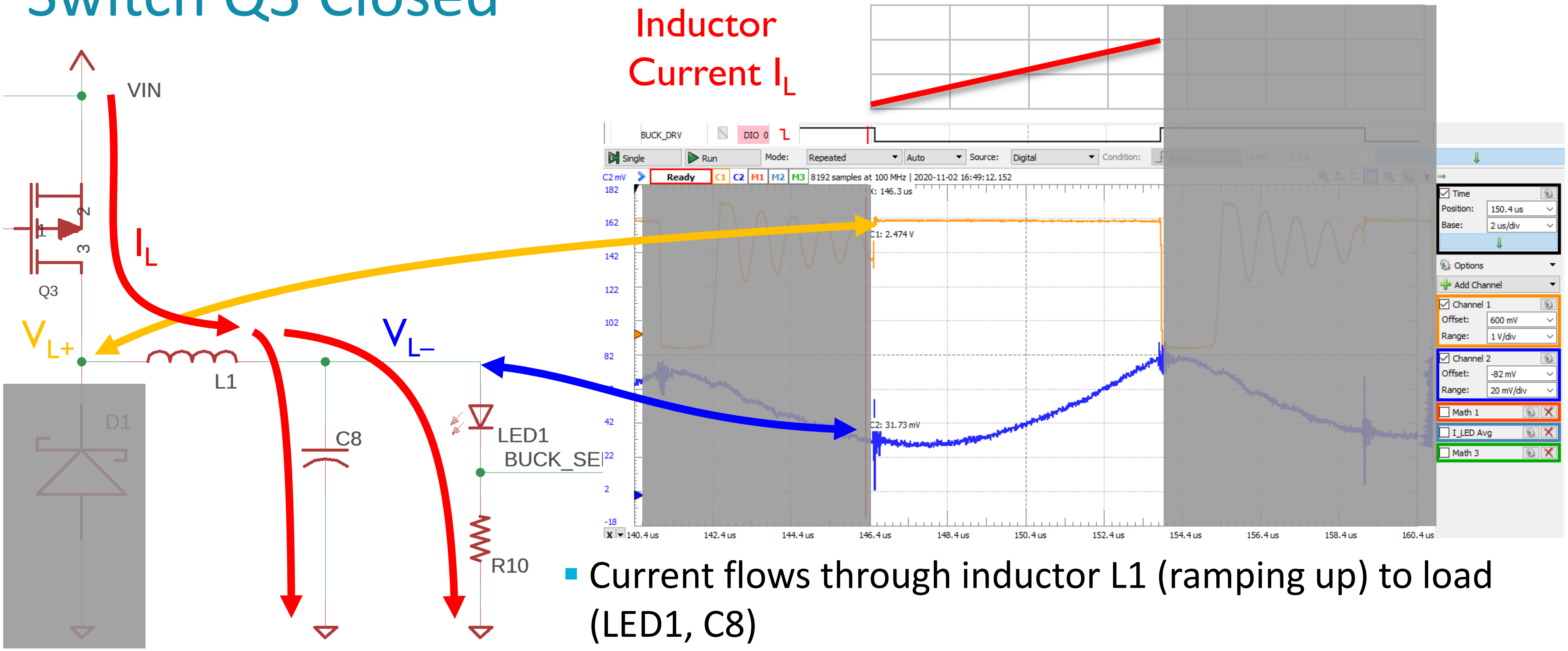
- Convert energy between electrical and/or magnetic forms for efficient storage and filtering
- Use capacitors and inductors to store energy and filter out ripple
  - Capacitors – **voltage** is stabilized
  - Inductors – **current** is stabilized
- Capacitor stores energy as voltage difference
  - $E = \frac{1}{2} CV^2$
  - Total storage limited by voltage: need higher voltage
  - Current affects charge/discharge rate, but not total energy storage
- Inductor stores energy as current and magnetic field
  - $E = \frac{1}{2} LI^2$
  - Total storage limited by current: need higher current
  - Voltage affects charge/discharge rate, but not total energy storage
  - Good match for low voltage operation. Easier to increase current capacity.

Buck Converter Overview



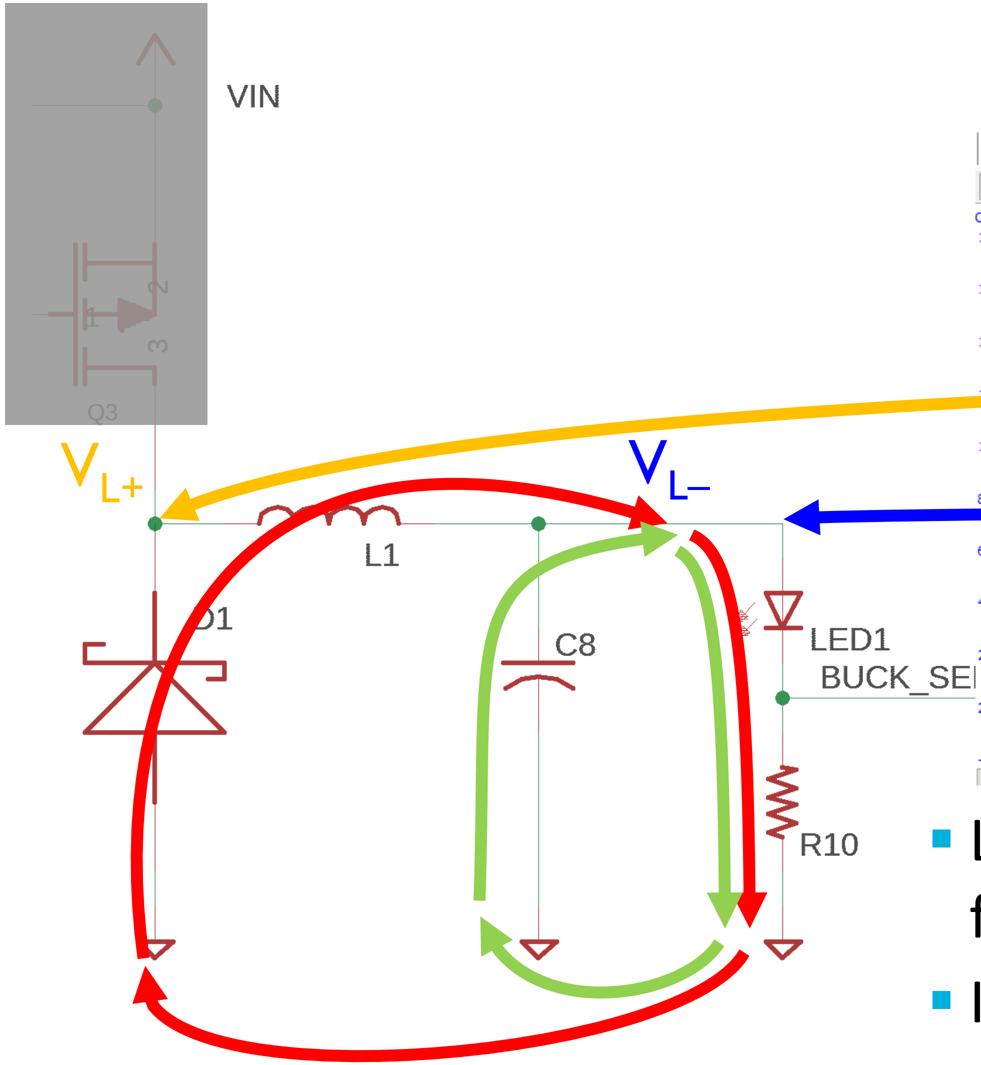
- PWM input signal
  - Duty cycle determines control effort
- Switches
  - Transistor Q3 is on when PWM signal on gate is active (low)
  - Diode D1 is off when Q3 is on (controlled by voltage at Q/D/L node)
- Can add capacitor C8 on output to store some energy

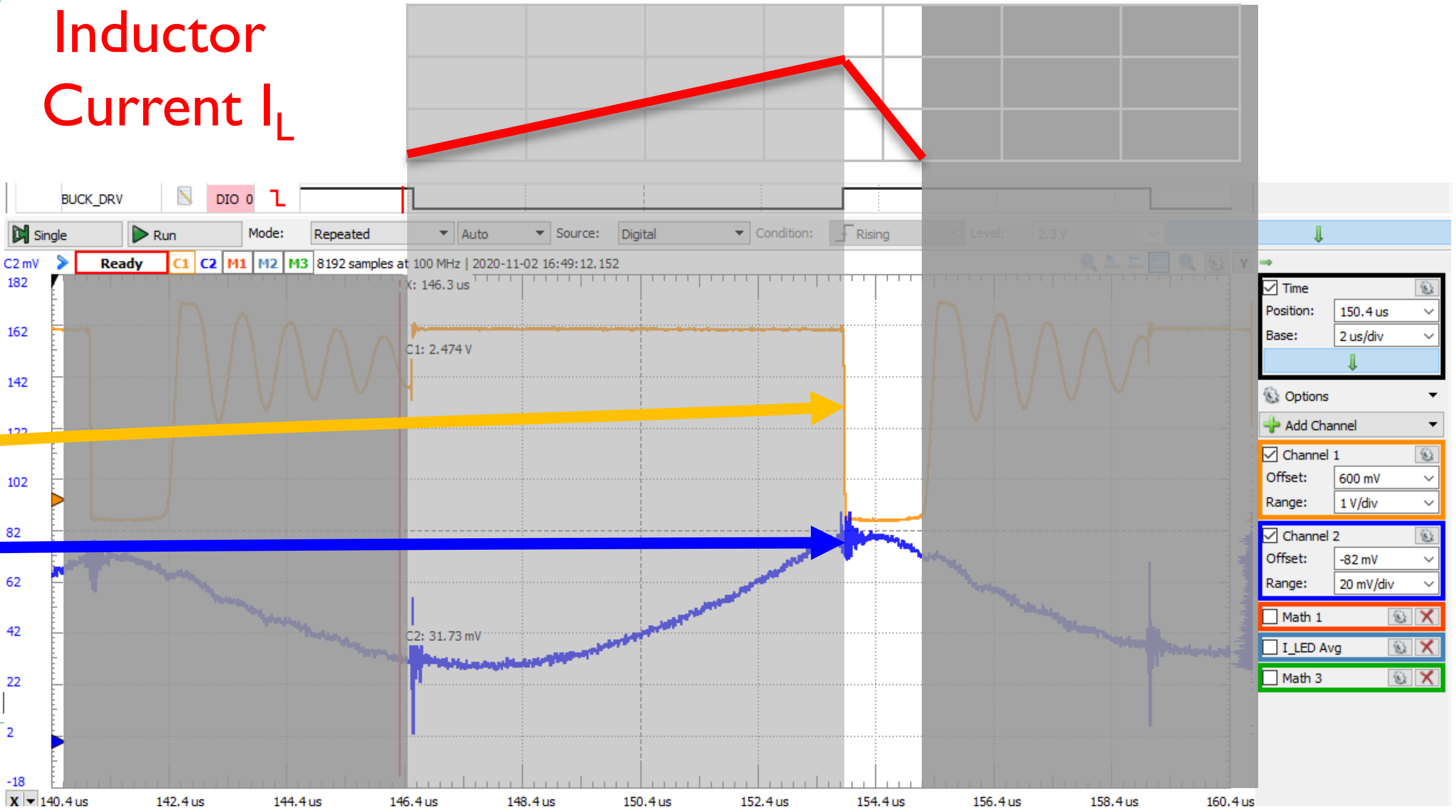
Switch Q3 Closed



- Current flows through inductor L1 (ramping up) to load (LED1, C8)
  - $di(t)/dt = (V_{L+} - V_{L-})/L$
  - Slope depends on voltage applied across inductor
- Charges up L1 (inductor creates magnetic field) and C8

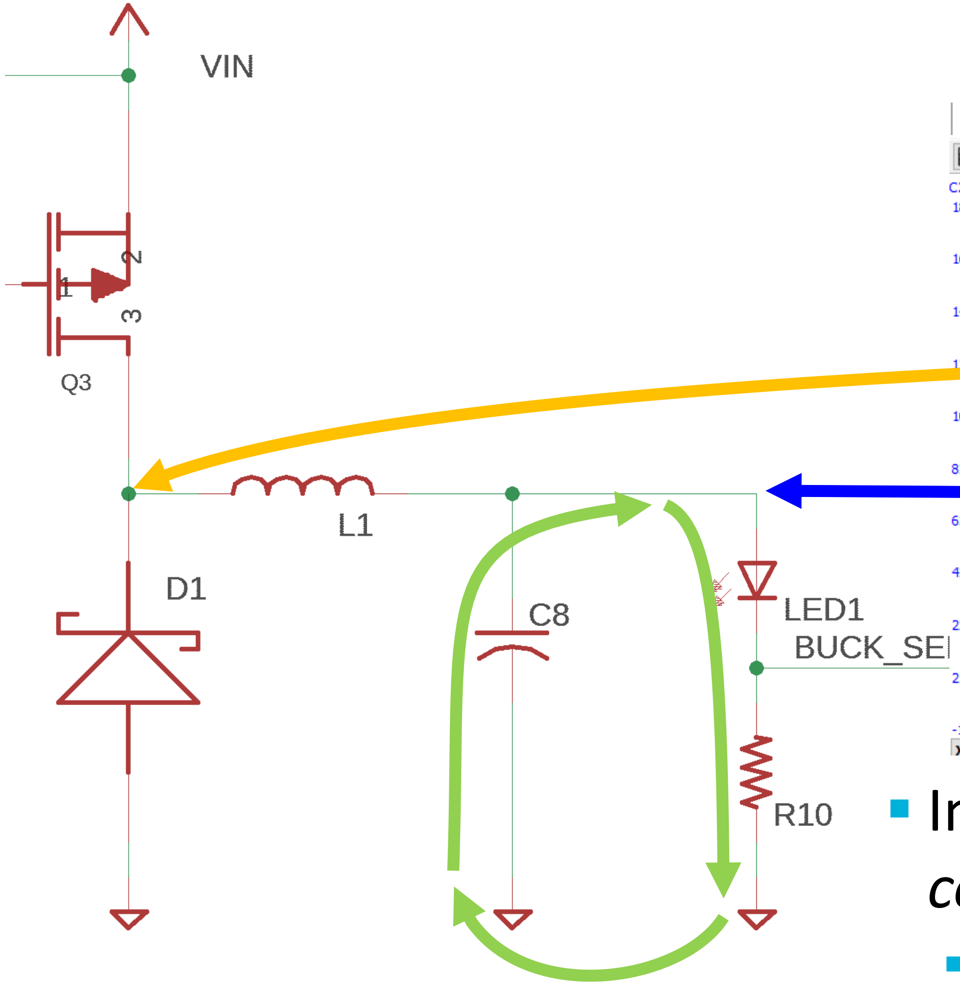
Switch Q3 Open, $I_L>0$

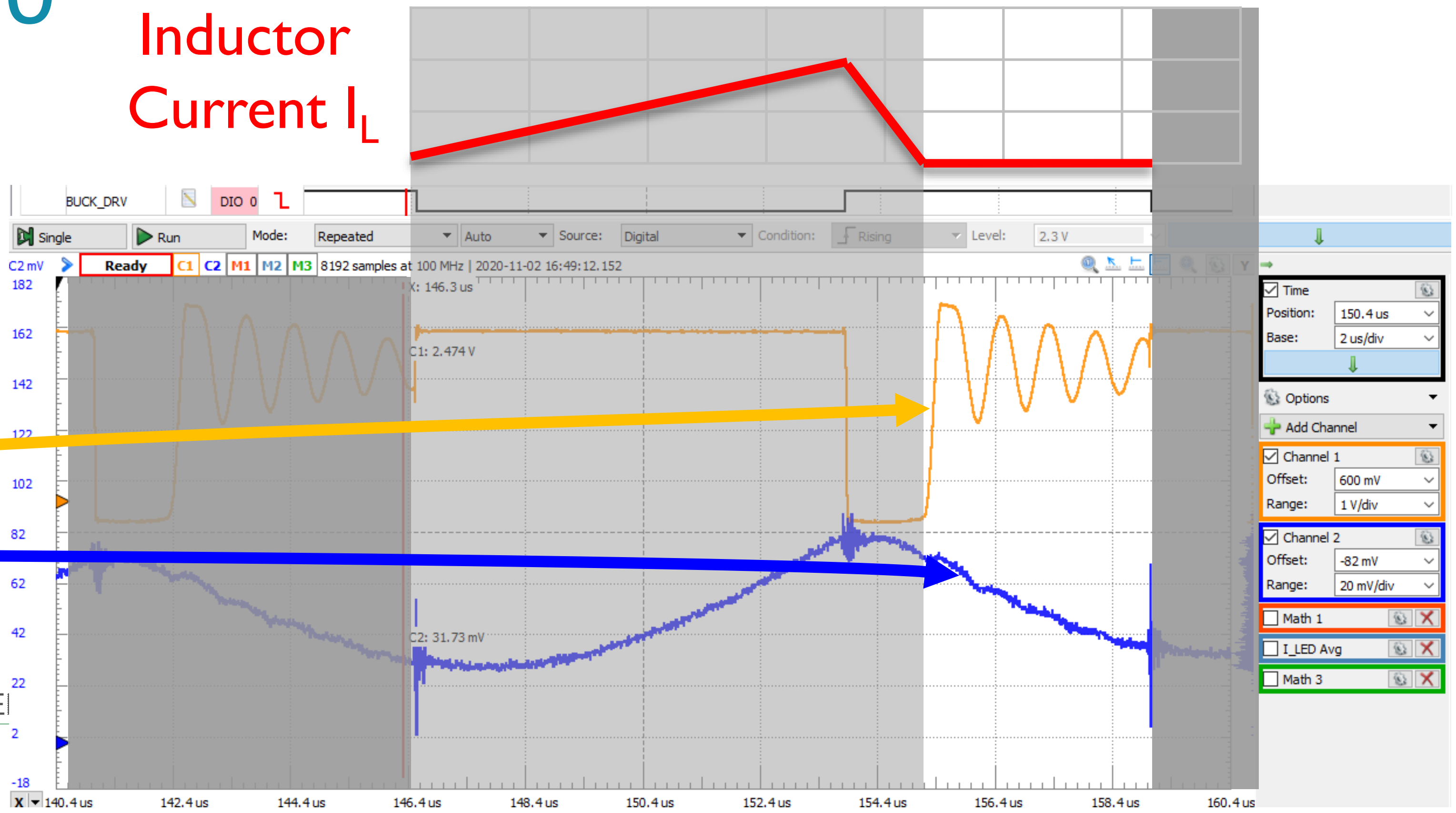




- Load (LED) is powered by current from L1 and C8 (for filtering
- Inductor current flows through D1 now, ramps down
  - Larger load current will make I_L ramp down faster

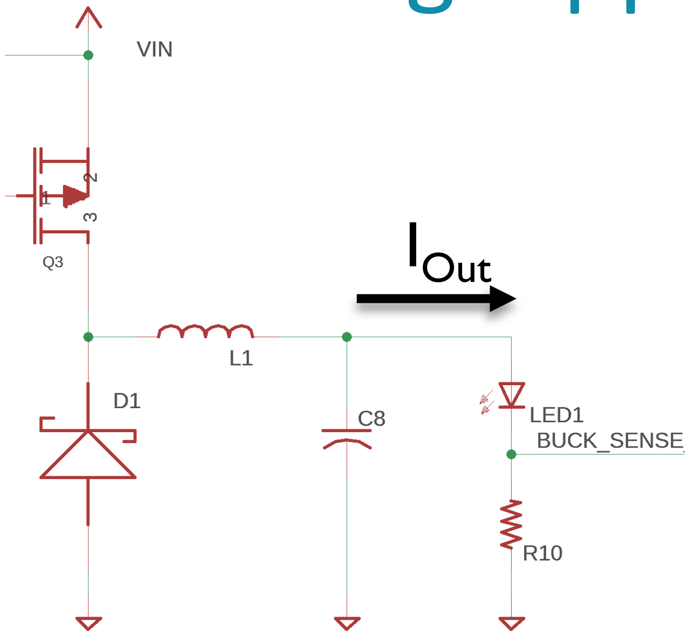
Switch Q3 Open, $I_L = 0$

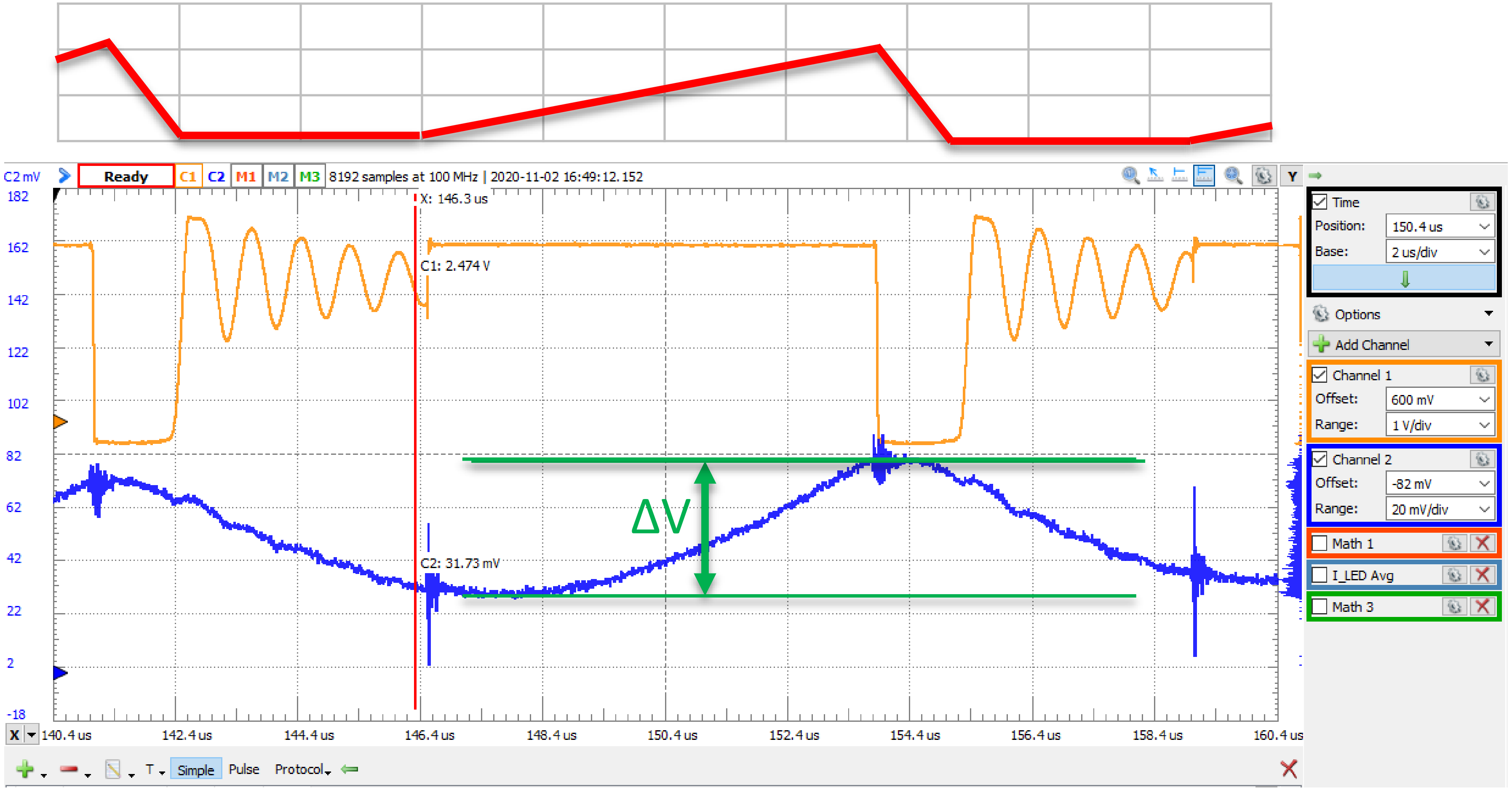




- Inductor current reaches 0, so now in *discontinuous conduction mode*
  - Models (equations) get more complex
- Load is powered by C8
- Oscillation from L1 and parasitic capacitances of D1 and Q3

Switching Ripple





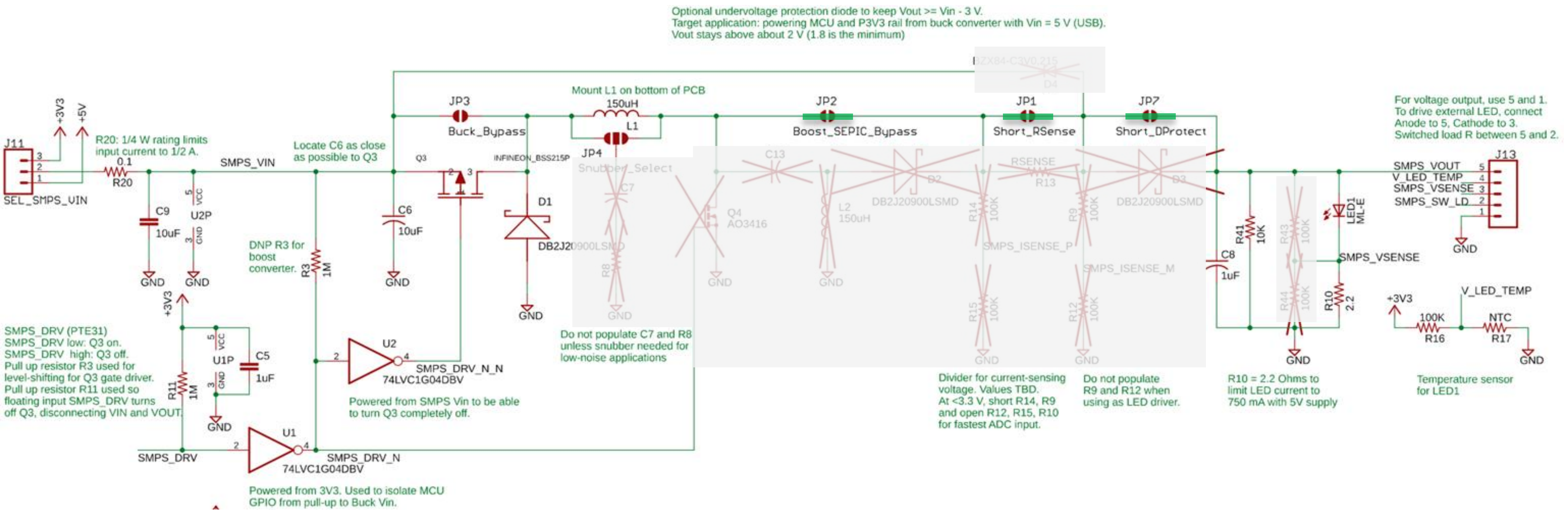
- Switching creates ripple in I_L,V_{Out}
- Output ripple voltage
  - Depends on load current I_{Out}, C8, switching frequency
  - $\Delta V = \frac{I_{Out}}{4Cf}$

Expansion Shield's Switch-Mode Power Converter

Shield Switch-Mode Power Converter Overview

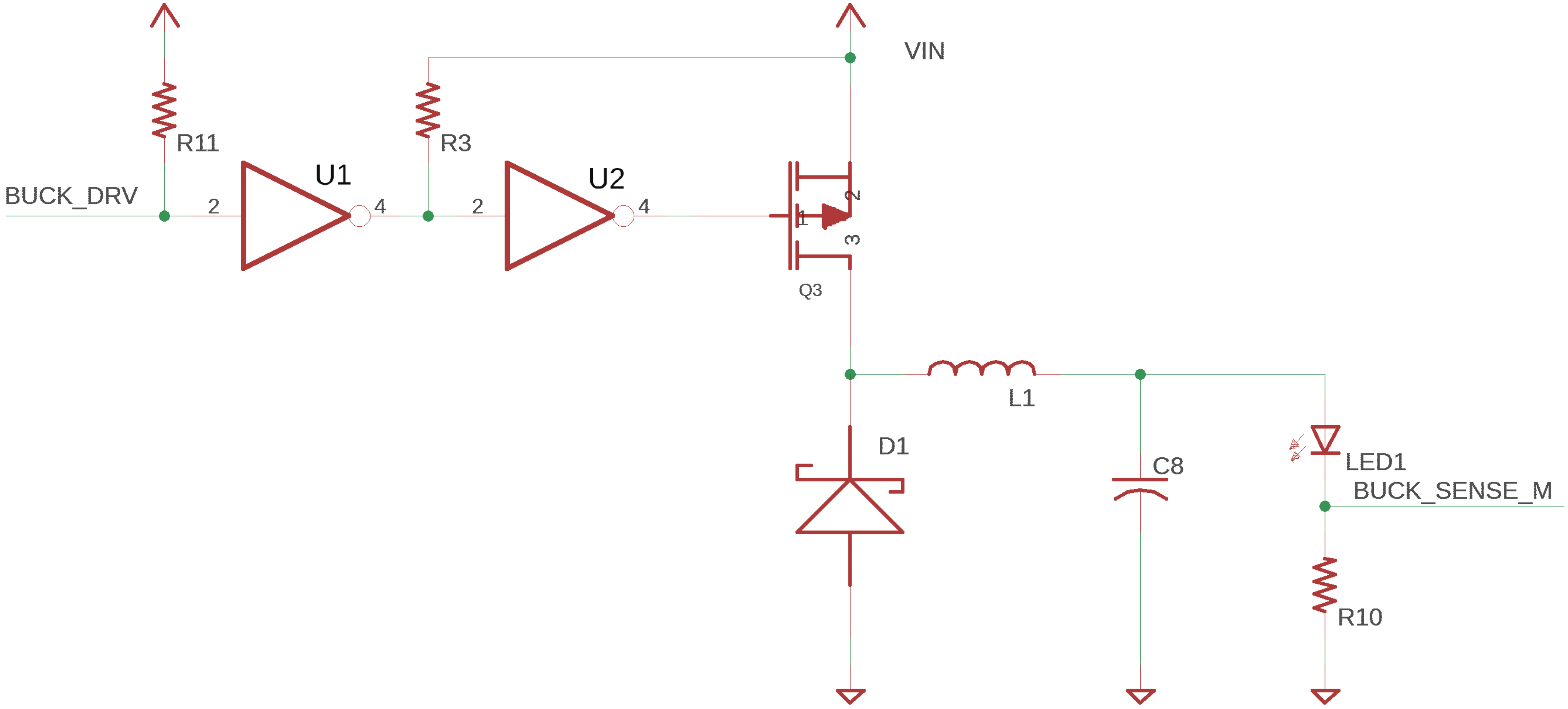
- Used to convert power efficiently one voltage to another
- Configurable hardware
  - Topology and function
    - Buck converter (reduce voltage) – we use this
    - Boost converter (increase voltage)
  - Noise control
    - Snubber (R8, C7) to reduce EMI when Q3 switches on, off
  - Switchable load transistor (Q2) for testing response to changes in load
  - Feedback options
    - Output voltage scaled by R9/(R9+R12). Single-ended (ground-referenced)
    - Current out through LED1, scaled by R10 (2.2 mV/mA). Single-ended (ground-referenced)
      - R10 selected to limit maximum current through LED1 to safer level
    - Current out before capacitor C8, scaled by sense resistor R13. Differential (not ground-referenced)

Schematic of SMPS Configured as Buck Converter

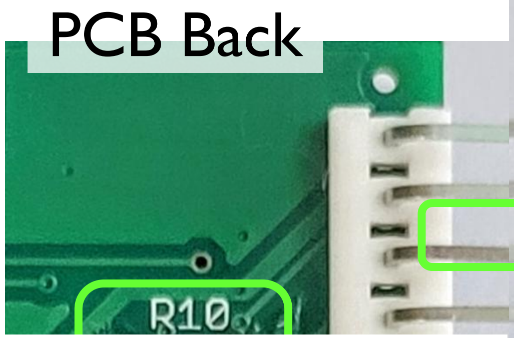


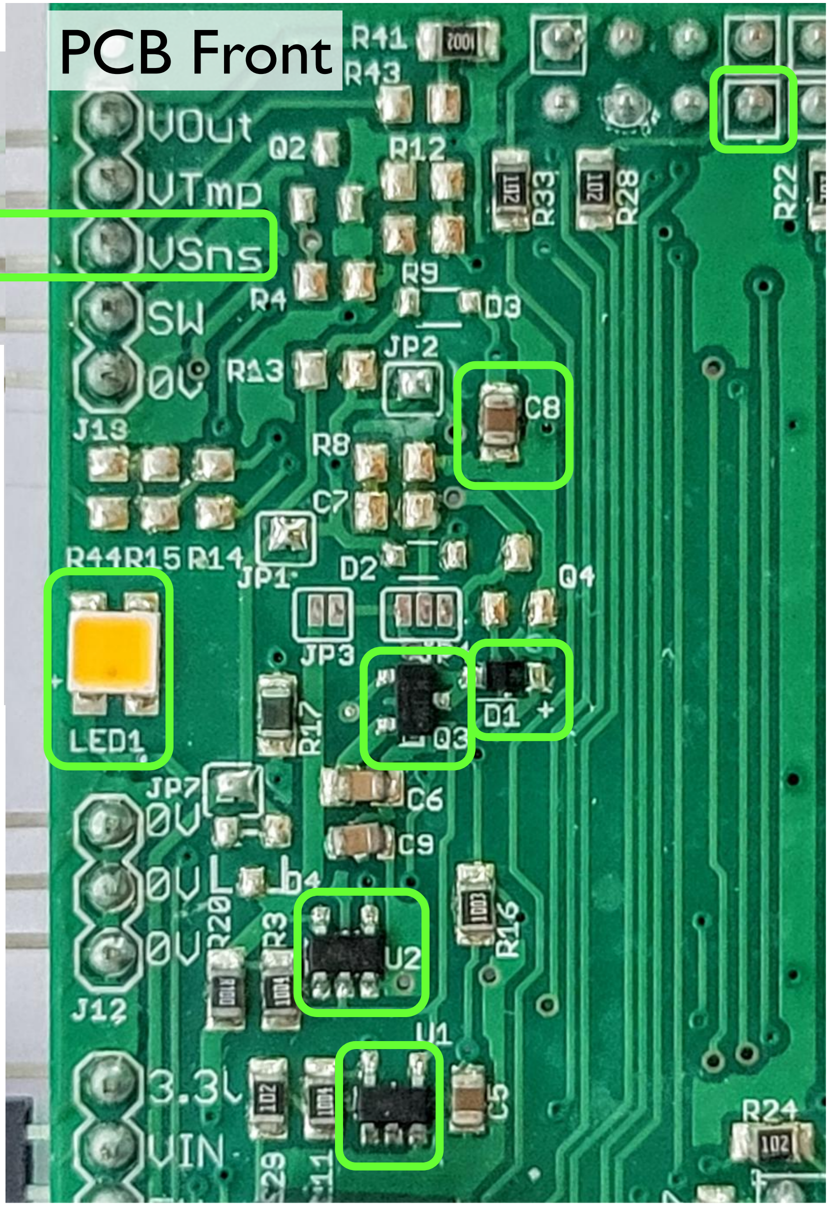
- Unused components are grayed or X'd out
- Shorted solder jumper

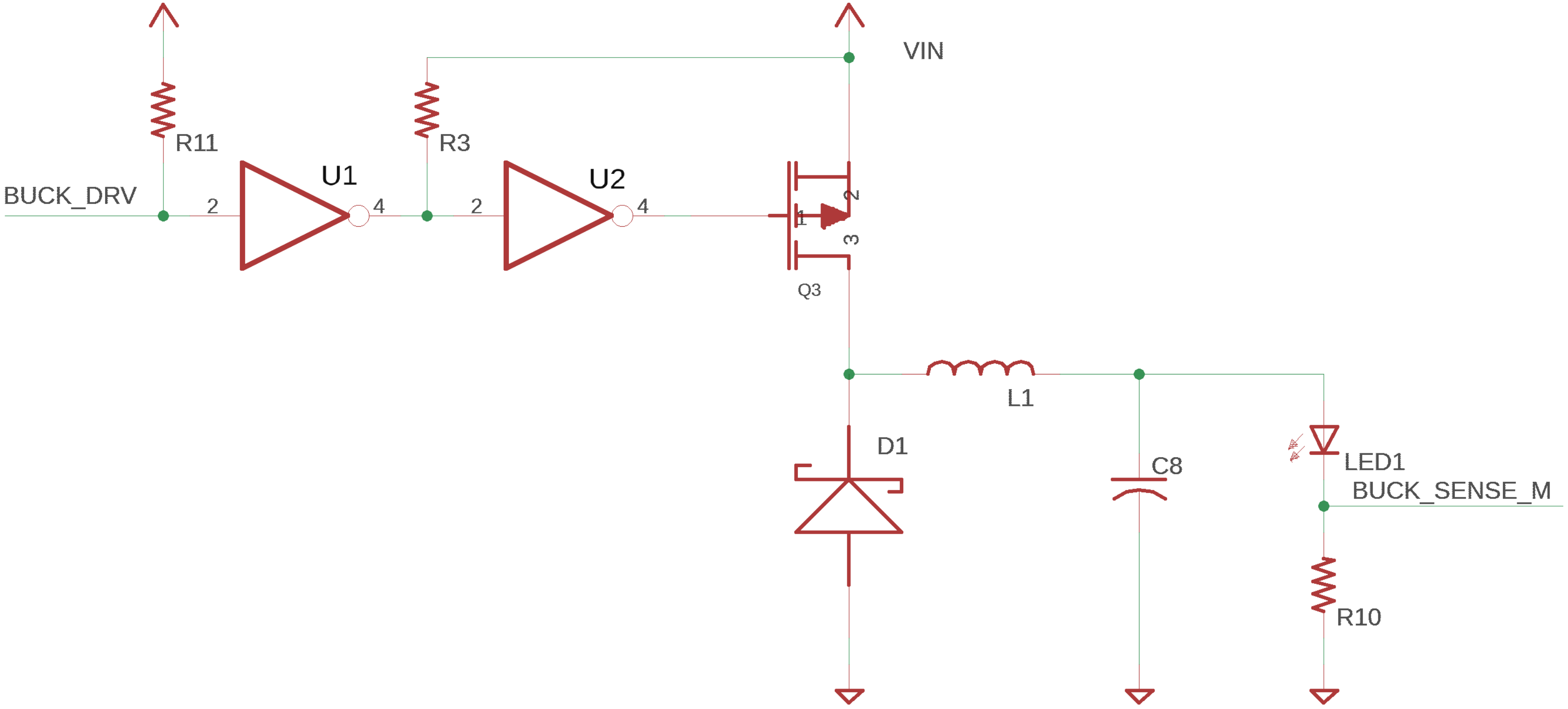
Simplified Buck Converter Schematic



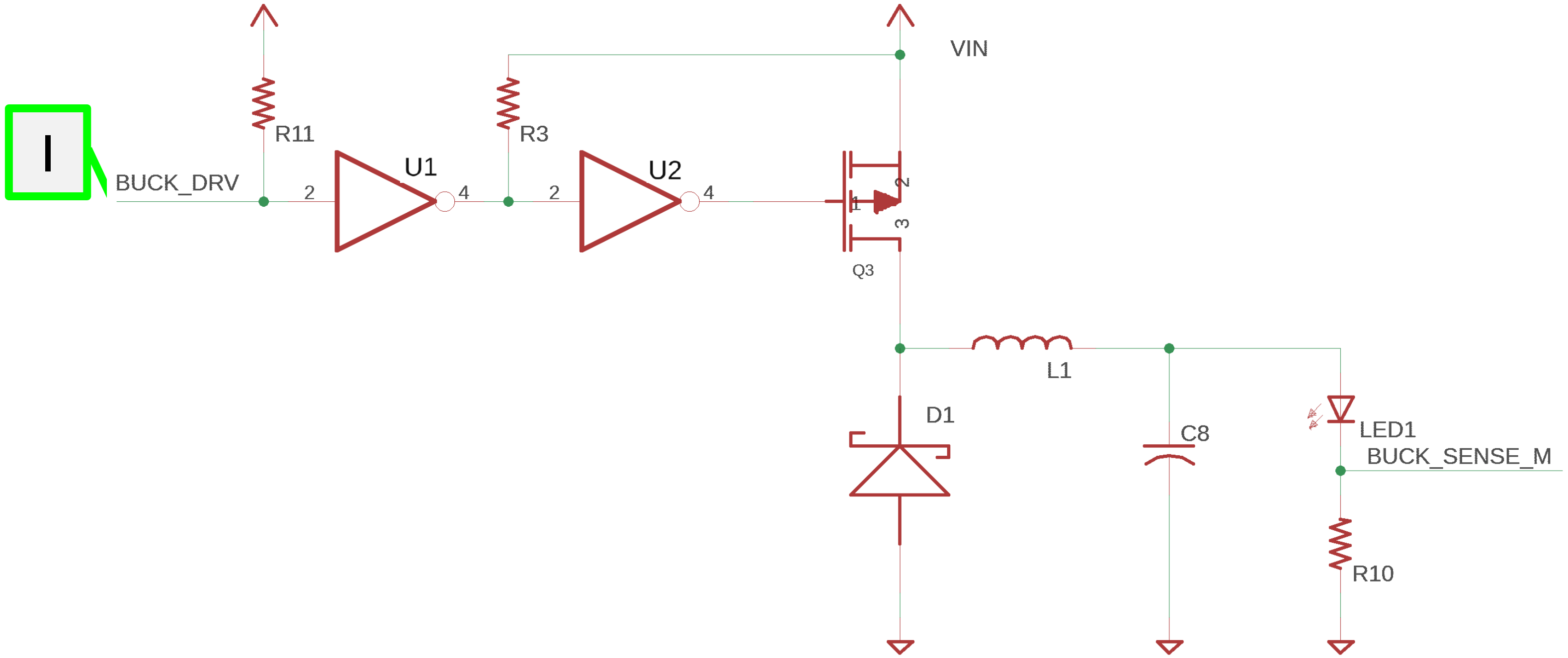
Buck Converter on PCB

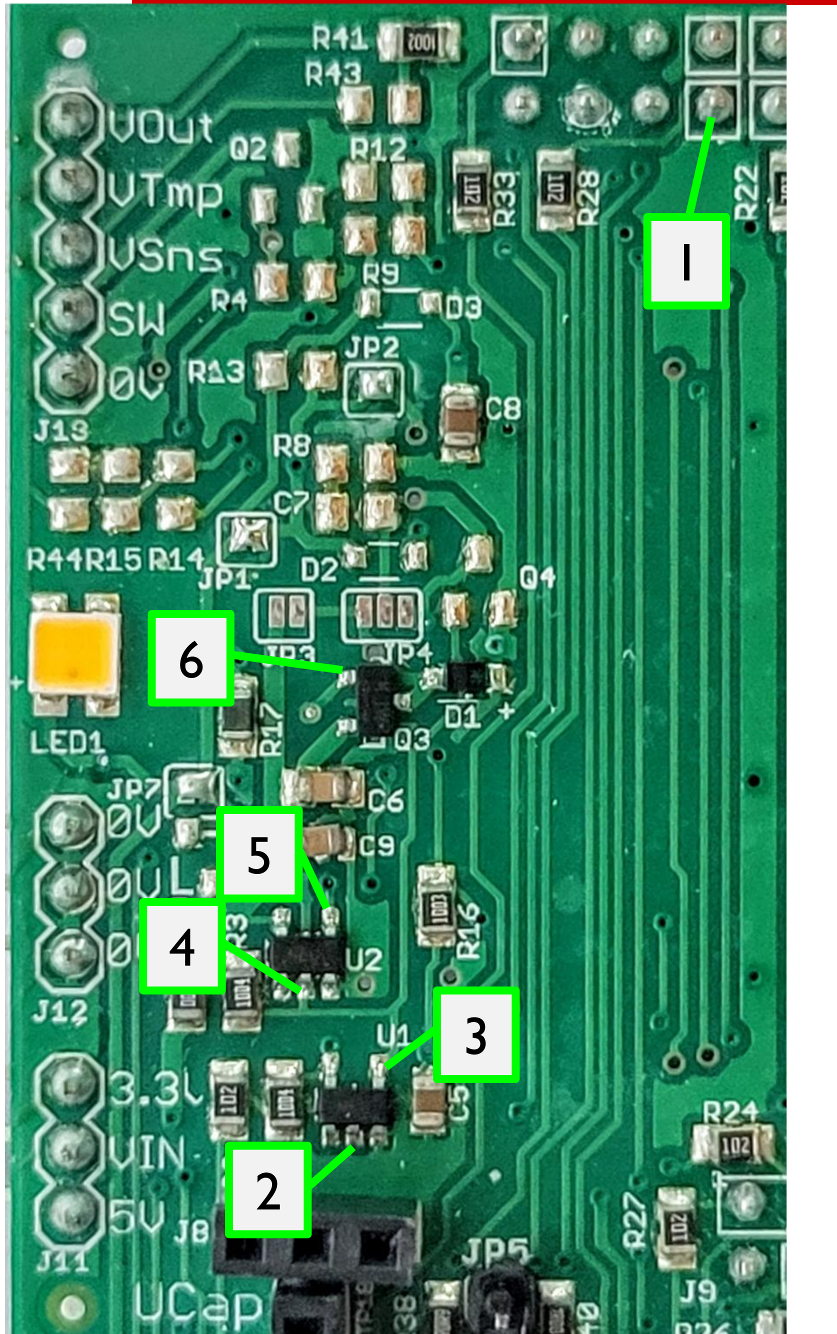






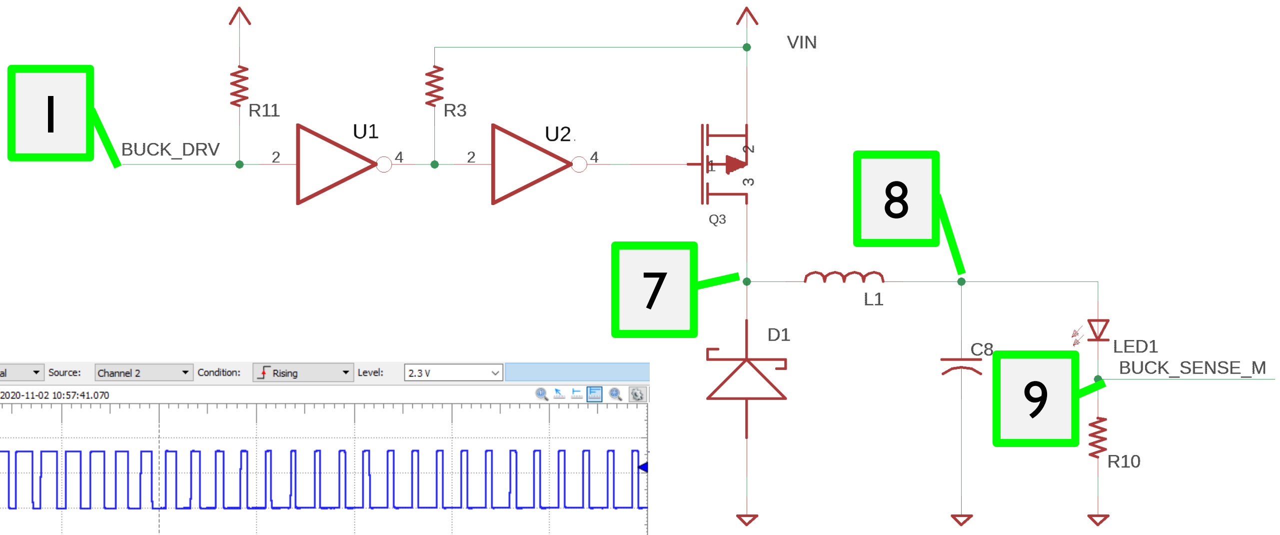
Tracing the BUCK_DRV PWM Signal

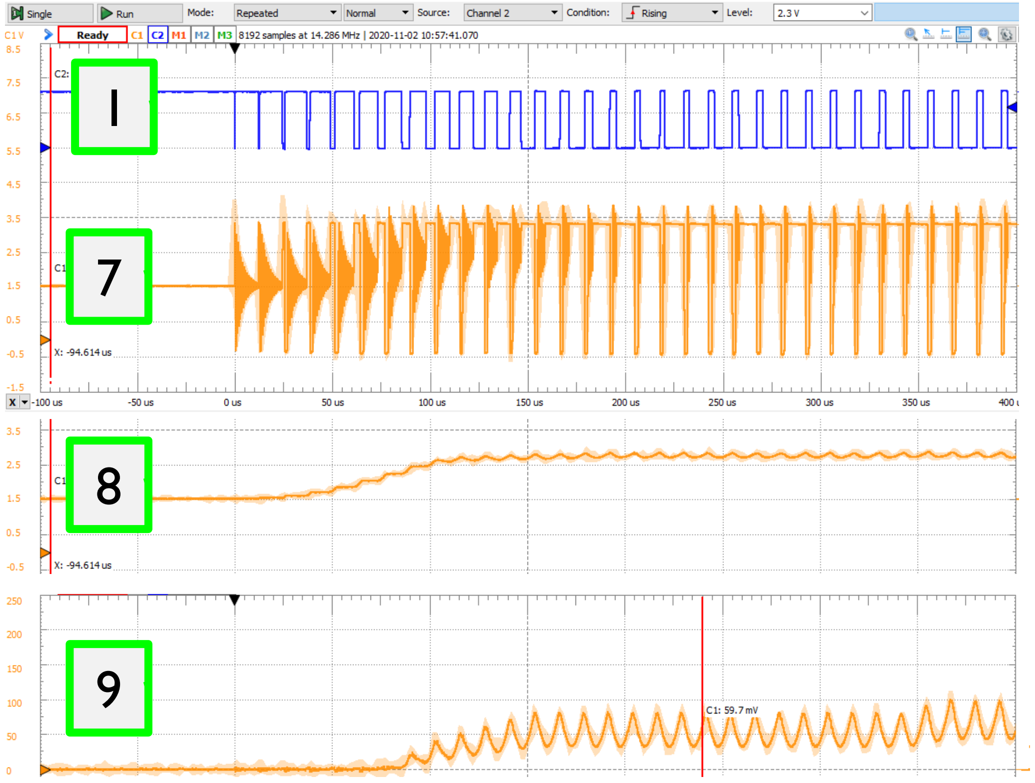


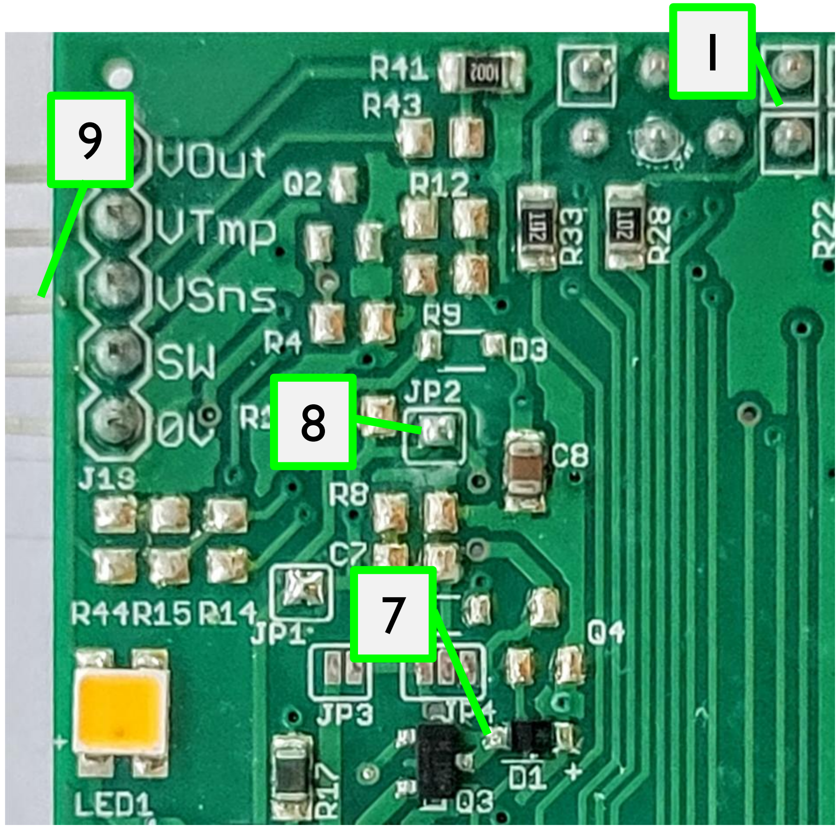


1. Confirm PWM signal at header pin
2. Confirm PWM signal at input to U3
3. Confirm inverted PWM signal at output of U3
4. Confirm inverted PWM signal at input to U1
5. Confirm PWM signal at output of U1
6. Confirm PWM signal at gate of Q3

Buck Converter Power Stage Signals

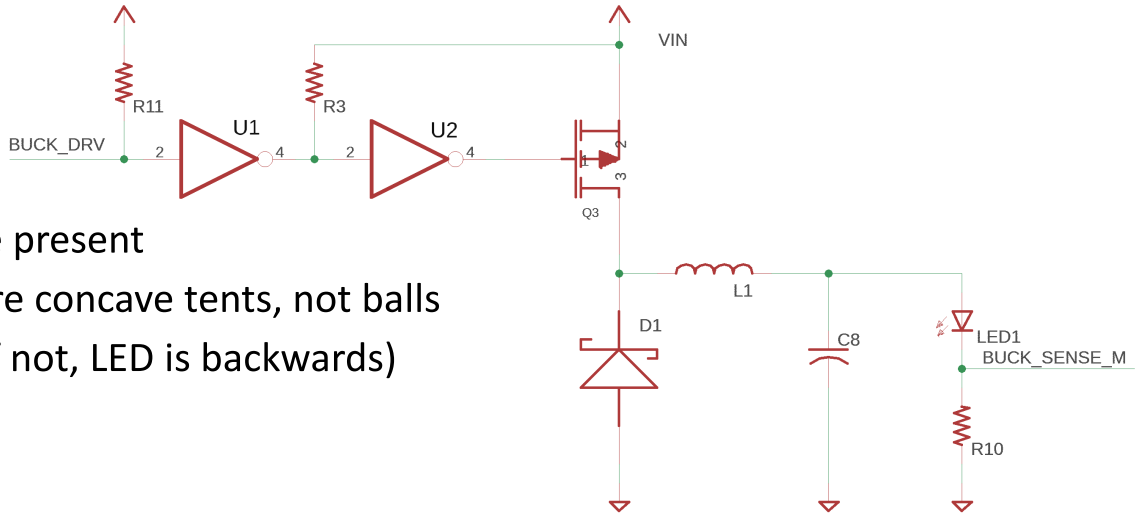




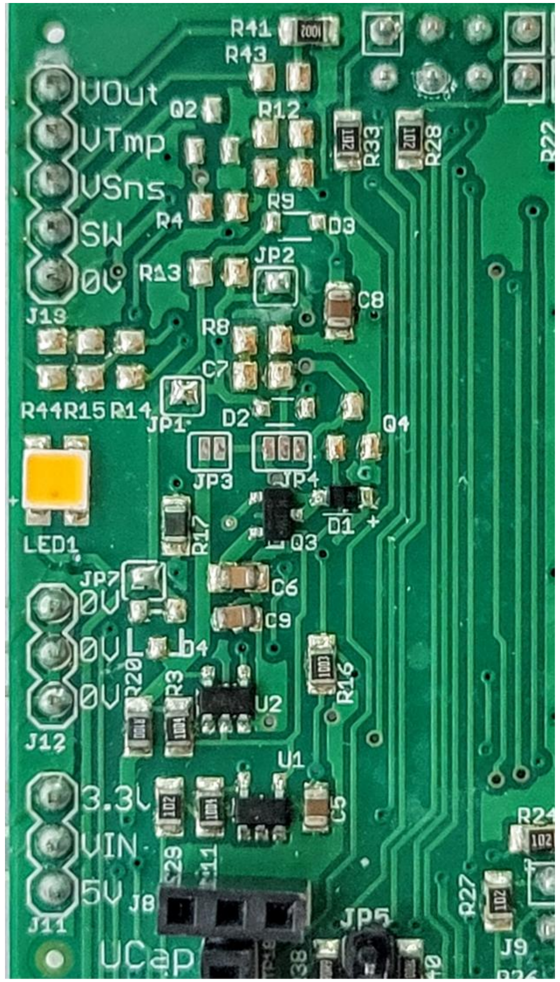


7. Confirm switching signal at Q3/D1/L1 node. 1 V/division
8. Confirm output voltage at JP2 (L1/C8/LED1 node). 1V/division. *As C8 not used, will have large ripple.*
9. Confirm current sense voltage at BUCK_SENSE_M (S-). 50 mV/division. *As C8 not used, will have large ripple.*

Visual, Resistance and Diode Tests – POWER OFF!



- Remove power and J11 jumper
- Visual checks. Confirm:
  - All components in this schematic are present
  - Solder connections to header pins are concave tents, not balls
  - Dot on LED1 (anode) is closer to + (if not, LED is backwards)
  - JP1 and JP2 are shorted
- Multimeter resistance tests
  - J13 S- to 0V should be 2-3 Ω. If larger, check for bad soldering on R10, J13
  - BUCK_VIN to ground should be large (>1M, rising)
  - Q3 pin 3 / D1 Anode / L1 node to ground should be large (>1M, rising)
  - JP1 to ground should be large (>1M, rising)
- Multimeter diode tests
  - D1 should show forward voltage of about 0.24 V
  - LED1 should show forward voltage of about 2.46 V
  - Q3
    - + on Gate (2), - on Drain (3) forward voltage 1 V
    - + on Drain (3), - on Gate (2) forward voltage 0.24 V



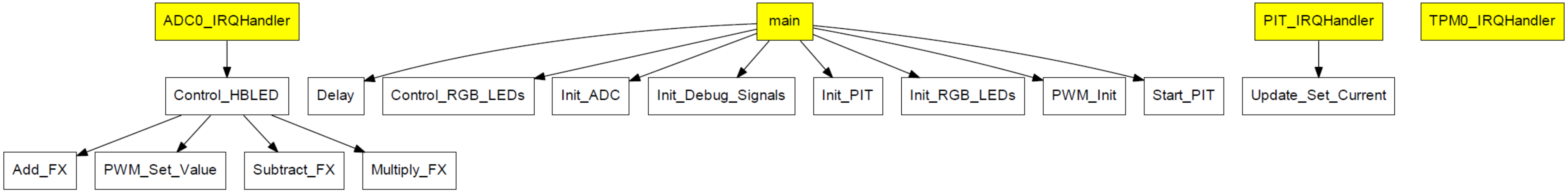
Architecture for Constant Current HBLED Driver using Buck Converter

Configuration in HBLED.h

- Control architectures available
  - *"Hard-Coded": Must rebuild code to change, since code for run-time changes would be too slow*
  - Asynchronous sampling
    - Free-running S/W loop in main triggers ADC
  - Synchronous sampling
    - No frequency division: $f_{ctl} = f_{sw} = f_{TPM0}$
      - ADC hardware triggered by TPM0 overflow
    - Software frequency division: $f_{ctl} = f_{sw}/N = f_{TPM0}/N$
      - N set by SW_CTL_FREQ_DIV_FACTOR
      - ADC software triggered by TPM0 ISR
    - Dual timers: $f_{ctl} = f_{TPM2}$, $f_{sw} = f_{TPM0}$
      - TPM2 ISR enables TPM0 IRQ once
      - TPM0 ISR software triggers ADC, disables TPM0 IRQ
- Controllers (compensators) available
  - *"Software-Configurable": Can change compensator and gains by tweaking variables as program runs*
  - Types
    - None (open loop)
    - Bang-Bang
    - Incremental
    - PID
    - Fixed-point PID
  - Parameters
    - Gains are scaled by control loop period (faster update rate reduces the required gain)

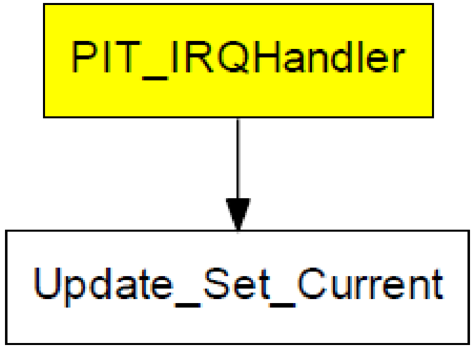
Which Functions Can Call Each Other?

Function Call Graph



- Functions in yellow are threads or interrupt handlers
- Calling behavior depends on the control architecture selected (USE_SYNC_SW_CTL_FREQ_DIV shown here)

Examining PIT_IRQHandler



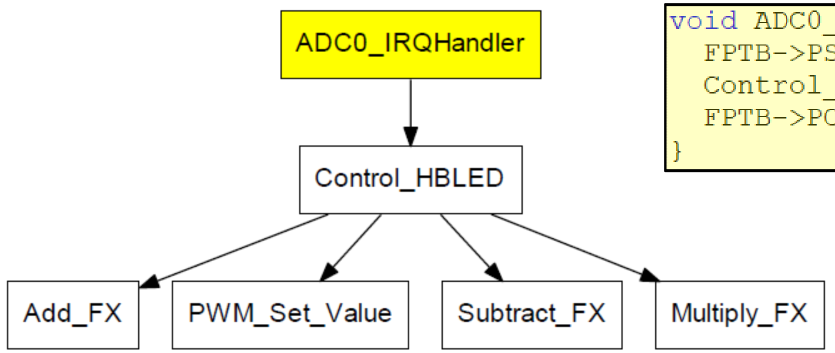

```c
void PIT_IRQHandler() {
  // check to see which channel triggered interrupt
  if (PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
    // clear status flag for timer channel 0
    PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
    // Do ISR work
    Update_Set_Current();
  } else if (PIT->CHANNEL[1].TFLG & PIT_TFLG_TIF_MASK) {
    // clear status flag for timer channel 1
    PIT->CHANNEL[1].TFLG &= PIT_TFLG_TIF_MASK;
  }
}
```

```c
void Update_Set_Current(void) {
  static int delay=FLASH_PERIOD;

  if (g_enable_flash){
    delay--;
    if (delay==1) {
      FPTB->PSOR = MASK(DBG_LED_ON);
      g_set_current = FLASH_CURRENT_MA;
      Set_DAC_mA(g_set_current);
    } else if (delay==0) {
      delay=FLASH_PERIOD;
      FPTB->PSOR = MASK(DBG_LED_ON);
      g_set_current = FLASH_CURRENT_MA/4;
      Set_DAC_mA(g_set_current);
    } else {
      FPTB->PCOR = MASK(DBG_LED_ON);
      g_set_current = 0;
      Set_DAC_mA(g_set_current);
    }
  }
}
```

```c
void Set_DAC_mA(unsigned int current) {
  unsigned int code = MA_TO_DAC_CODE(current);
  // Force 16-bit write to DAC
  uint16_t * dac0dat = (uint16_t *)&(DAC0->DAT[0].DATL);
  *dac0dat = (uint16_t) code;
}
```

- Global variable accesses
  - Reads g_enable_flash
  - Reads and writes g_set_current
- Peripheral access
  - Writes to the DAC
- *Note: compiler inlined call to Set_DAC_mA*

Examining ADC0_IRQ

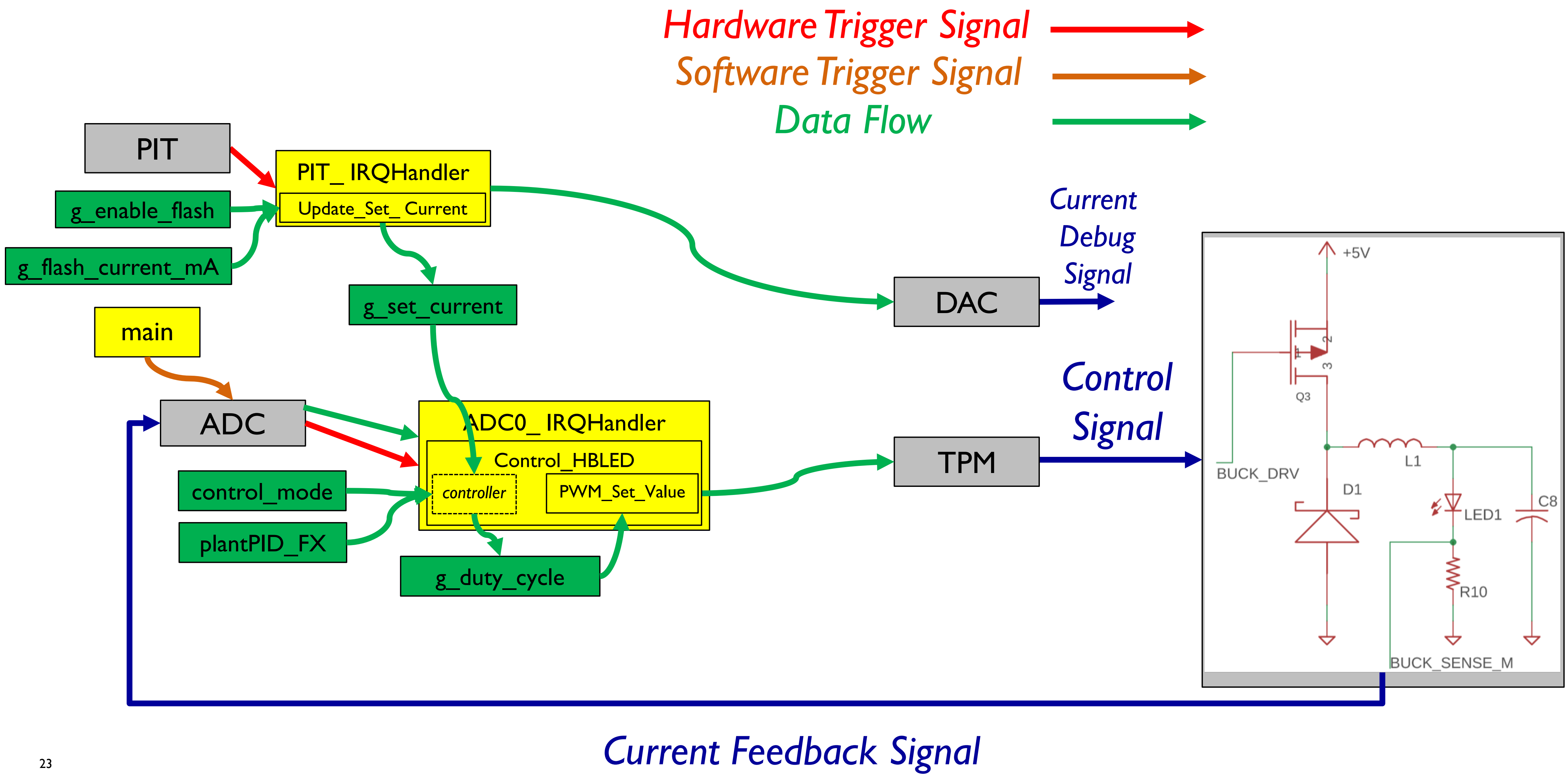



```
void ADC0_IRQHand
  FPTB->PSOR = MA
  Control_HBLED()
  FPTB->PCOR = MA
}
```

- Global variable accesses
  - Reads g_set_current
  - Reads and writes g_duty_cycle
- Peripheral access
  - Reads ADC0 result register
  - Writes g_duty_cycle to TPM0

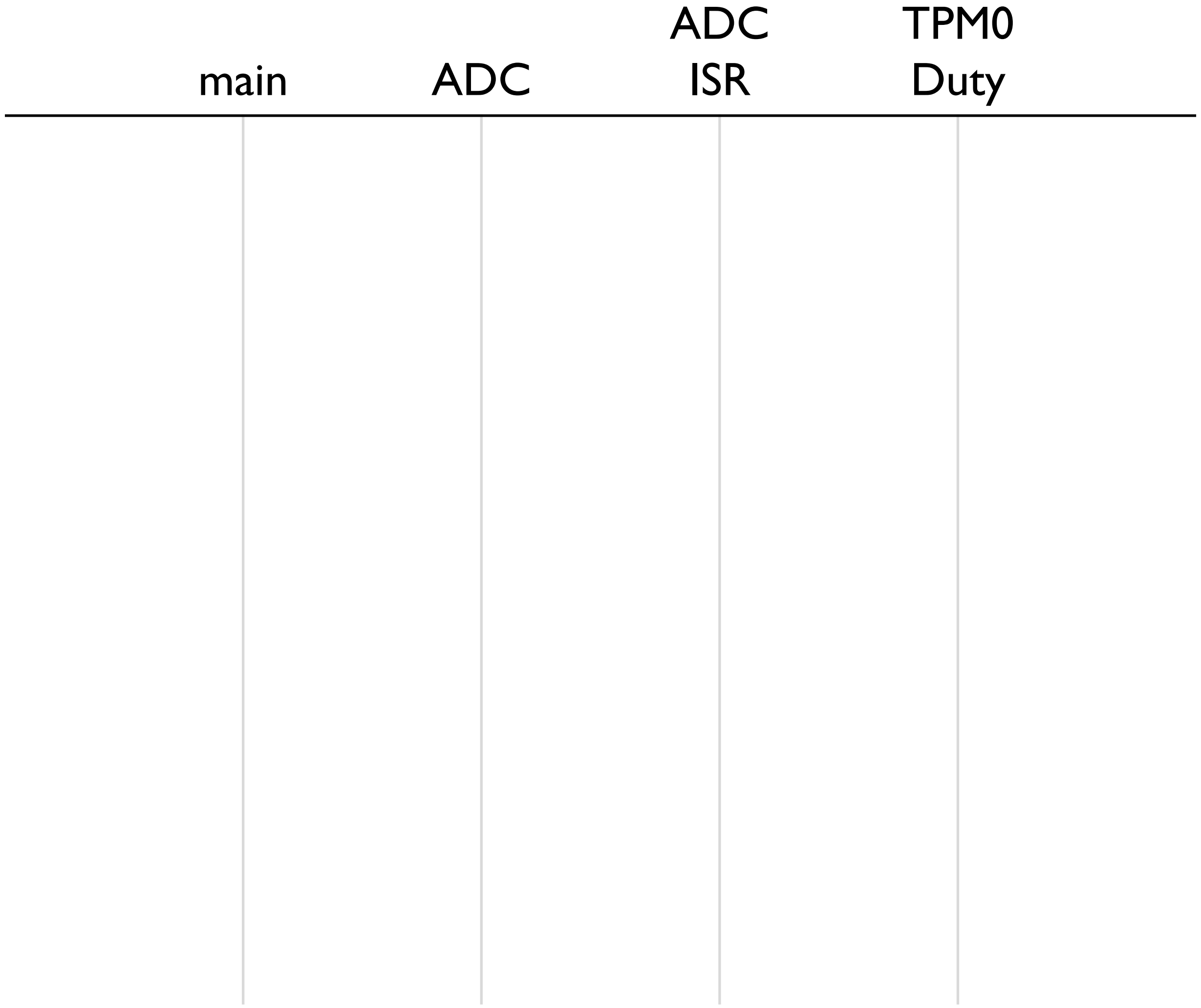
```
    ; // wait until end of conversion
#endif
  res = ADC0->R[0];

  measured_current = (res*1500)>>16; // Extra Credit: Make this code work: V_REF_MV*MA_SCALING_F

  switch (control_mode) {
    case OpenLoop:
        // don't do anything!
      break;
    case BangBang:
      if (measured_current < g_set_current)
        g_duty_cycle = LIM_DUTY_CYCLE;
      else
        g_duty_cycle = 0;
      break;
    case Incremental:
      if (measured_current < g_set_current)
        g_duty_cycle += INC_STEP;
      else
        g_duty_cycle -= INC_STEP;
      break;
    case Proportional:
      g_duty_cycle += (pGain_8*(g_set_current - measured_current))/256; //  - 1;
    break;
    case PID:
      g_duty_cycle += UpdatePID(&plantPID, g_set_current - measured_current, measured_current);
      break;
    case PID_FX:
      error_FX = INT_TO_FX(g_set_current - measured_current);
      change_FX = UpdatePID_FX(&plantPID_FX, error_FX, INT_TO_FX(measured_current));
      g_duty_cycle += FX_TO_INT(change_FX);
    break;
    default:
      break;
  }

  // Update PWM controller with duty cycle
  if (g_duty_cycle < 0)
    g_duty_cycle = 0;
  else if (g_duty_cycle > LIM_DUTY_CYCLE)
    g_duty_cycle = LIM_DUTY_CYCLE;
  PWM_Set_Value(TPM0, PWM_HBLED_CHANNEL, g_duty_cycle);
  FPTB->PCOR = MASK(DBG_CONTROLLER);
}
```

Asynchronous Sampling



ASYNC Sequence Diagram

USE_ASYNC_SAMPLING

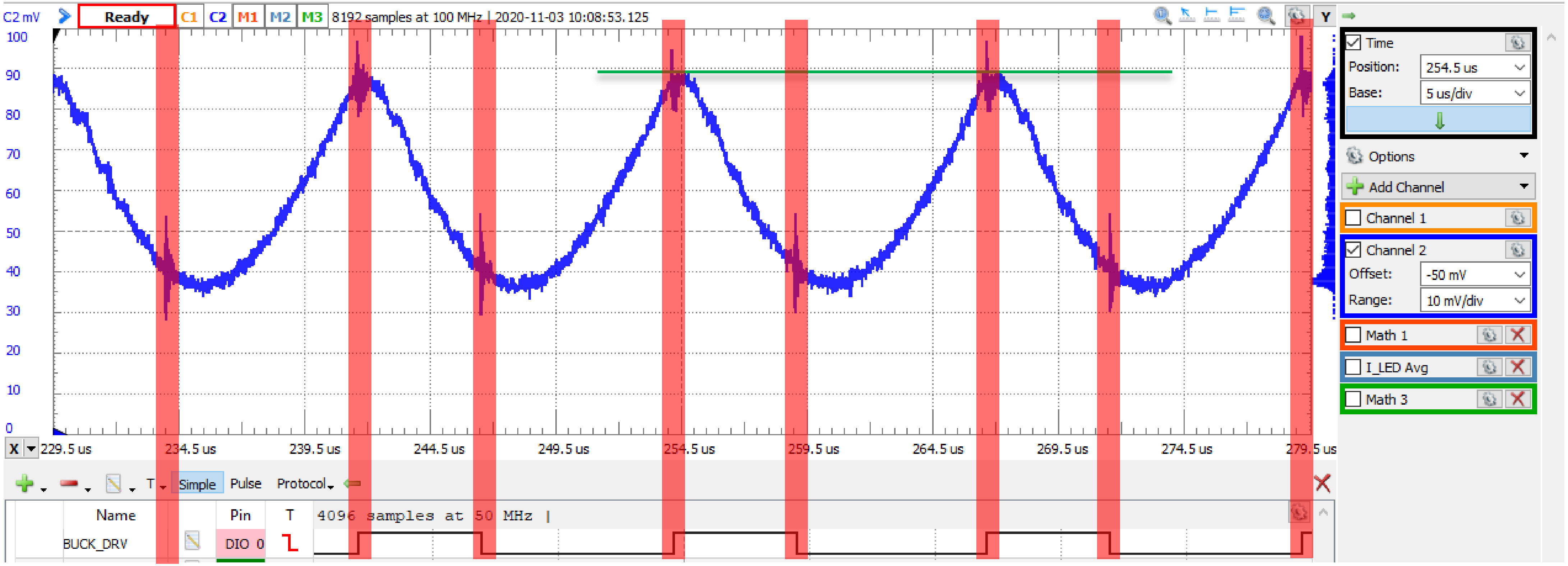


Timing Questions

- How long does it take to do the whole control loop?
  - How long does each individual part take?
  - How stable is the timing?

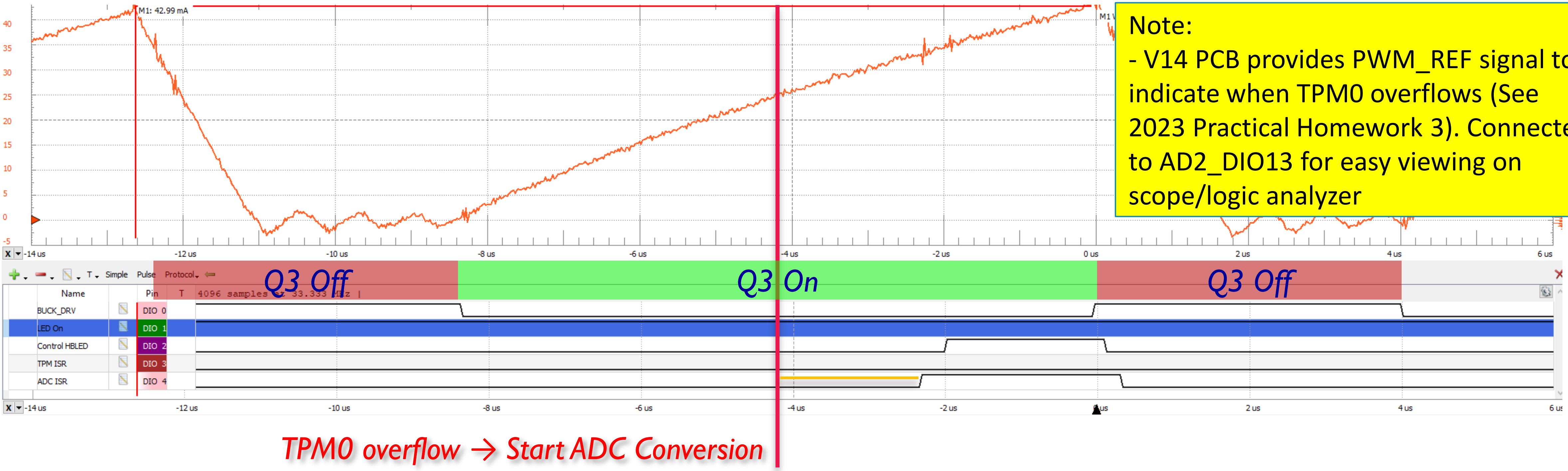
- When do we *really* sample the input voltage?
  - Can lead to under-estimate or over-estimate

When to Sample the LED Current?



- Need to sample in phase with BUCK_DRV signal (PWM)
  - Same phase for each sample, otherwise may have error of up to ΔV in each sample
  - Trigger with TPM
- Stay away from times of BUCK_DRV transitions (noisy!)
- Edge-Aligned vs. Center-Aligned PWM

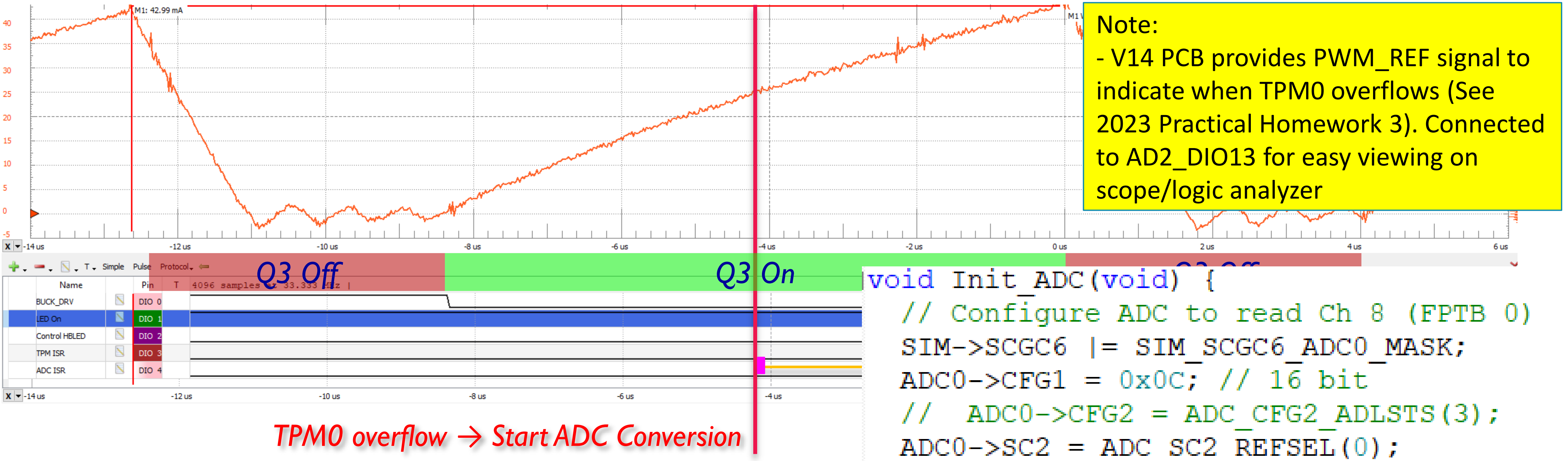
Examining Sampling Timing with Instrumentation



Note:
- V14 PCB provides PWM_REF signal to indicate when TPM0 overflows (See 2023 Practical Homework 3). Connected to AD2_DIO13 for easy viewing on scope/logic analyzer

- Signals monitored
  - Current feedback shows current, noise
  - BUCK_DRV shows when transistor is on, off
  - ADC_ISR shows when ISR runs
- When does ADC sample the analog input?
  - Starts a fixed time before the ADC_ISR runs (assuming no interference from other interrupts)
  - ADC conversion triggered by TPM0 overflow, which is in middle of transistor on time (BUCK_DRV low)

Examining Sampling Timing with Instrumentation

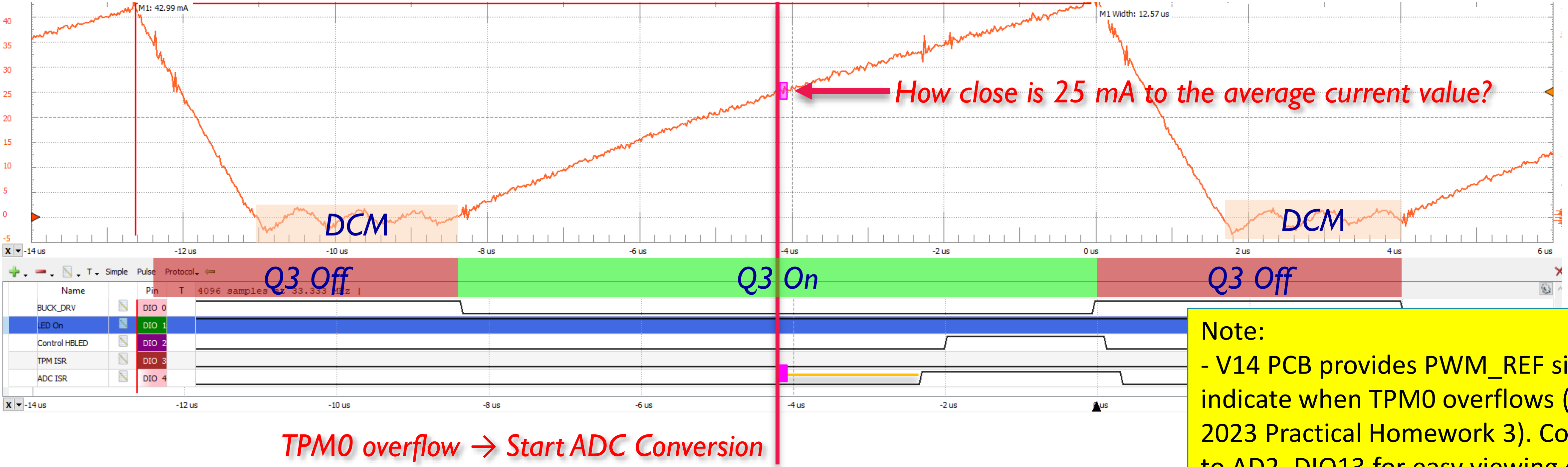



```
void Init_ADC(void) {
  // Configure ADC to read Ch 8 (FPTB 0)
  SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
  ADC0->CFG1 = 0x0C; // 16 bit
  //  ADC0->CFG2 = ADC_CFG2_ADLSTS(3);
  ADC0->SC2 = ADC_SC2_REFSEL(0);
```

- How long is the sample time? Check source code, refer to MCU manual section on ADC
  - Sample time is 4 ADCK cycles
  - What is ADCK?
    - 24 MHz (bus clock, no division)
  - Sample time = 4/24 MHz = 166 ns
- BTW, ADCK is too fast to meet accuracy specifications (max 19 MHz)!
  - Should set ADIV to 1 and ADHSC to 1 for 12 MHz ADCK

NC STATE UNIVERSITY

Examining Sampling Timing with Instrumentation



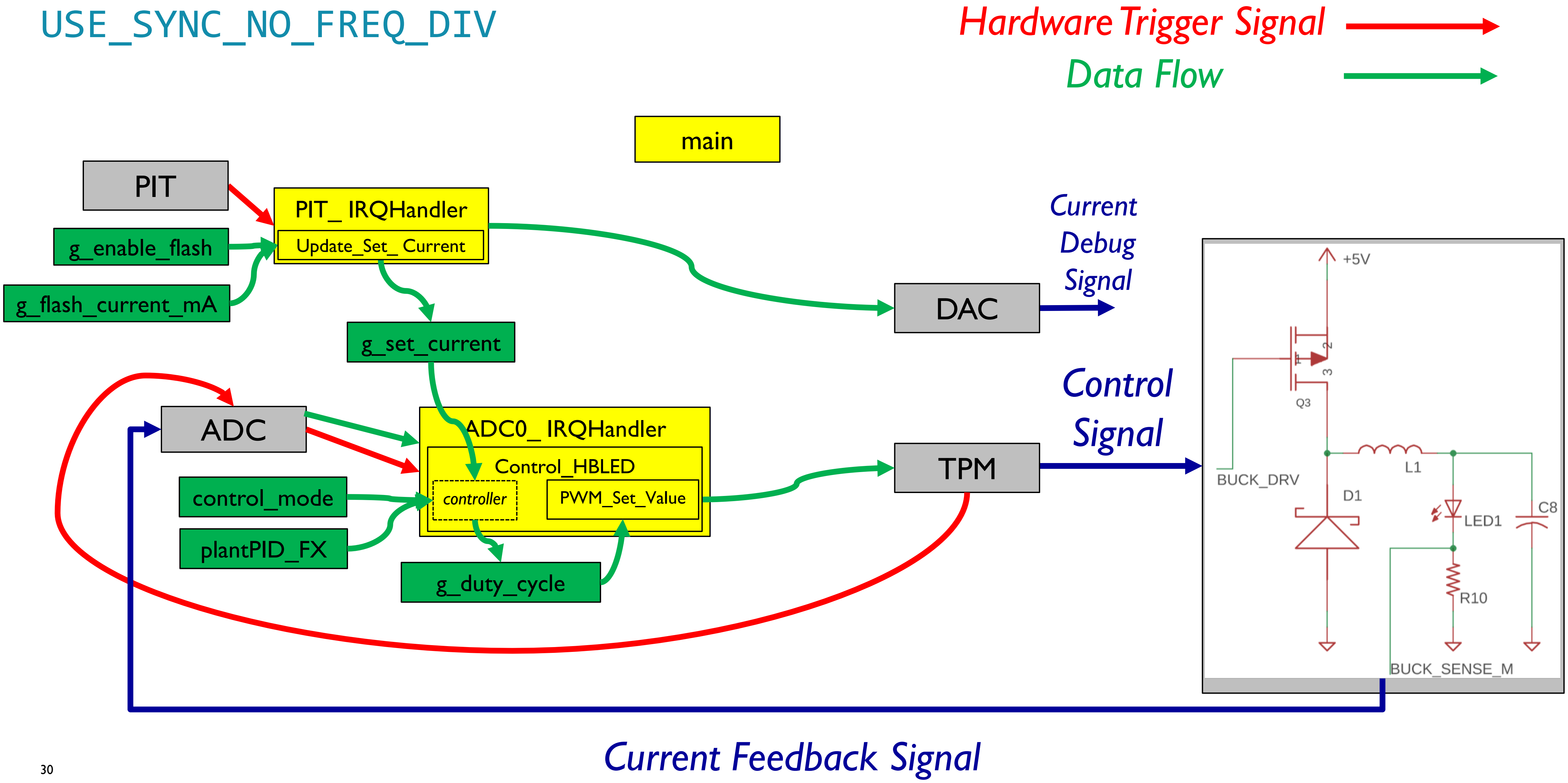
Note:
- V14 PCB provides PWM_REF signal to indicate when TPM0 overflows (See 2023 Practical Homework 3). Connected to AD2_DIO13 for easy viewing on scope/logic analyzer

- ADC sample and hold circuit holds signal 166 ns after start of conversion
  - I_{LED} = 25 mA at that point in time
- What is average value of this waveform?
  - Estimate graphically: (area of triangle/area of full period rectangle) * peak current ≈ 18 mA

- Error: 25 mA vs. 18 mA
  - Overestimates current by 39%
- Addressing error
  - Adding filtering capacitor (C8) would reduce this
  - Could scale reading, but depends on time spent in DCM (current oscillates around 0 mA). Need to evaluate stability of scaling across different duty cycles and load currents

Synchronous Sampling, No Frequency Division

USE_SYNC_NO_FREQ_DIV



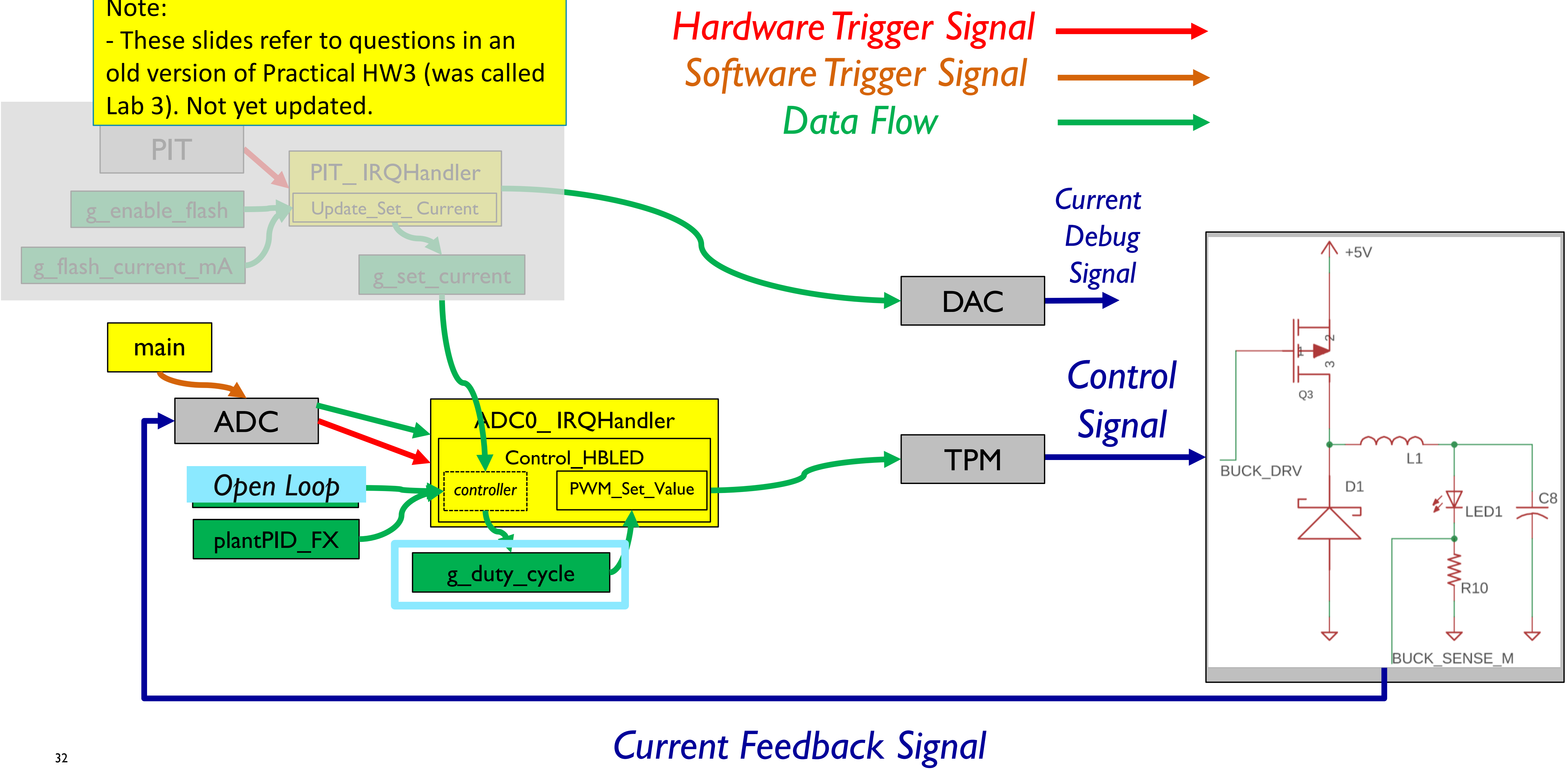
SYNC Sequence Diagram, No Division for F_{ctl}

USE_SYNC_NO_FREQ_DIV

TPM0 (f_{sw})	ADC	ADC ISR	TPM0 Duty

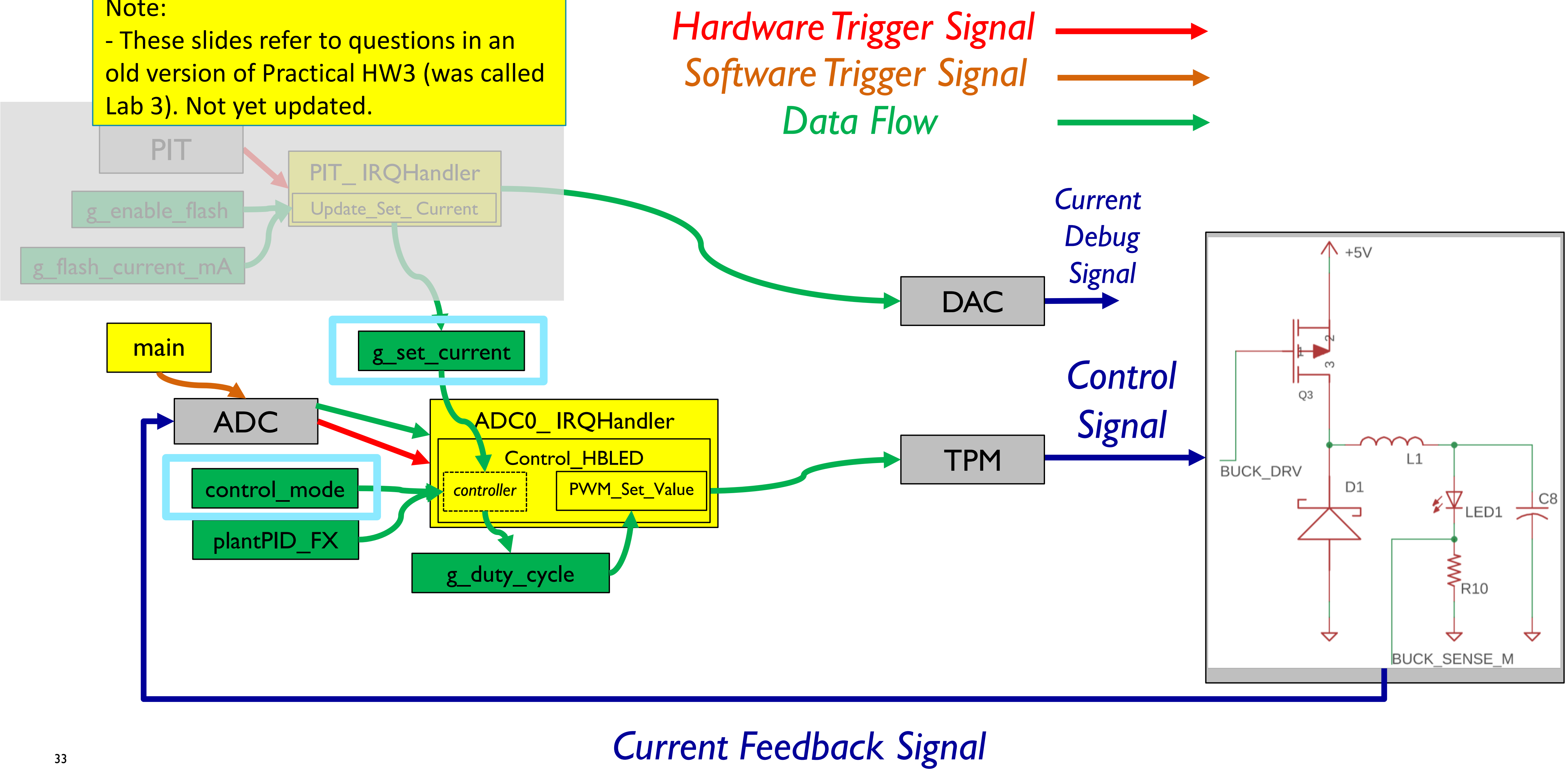
Lab 3: Q. 3-5 (Asynch. Sampling, Open-Loop)

Note:
- These slides refer to questions in an old version of Practical HW3 (was called Lab 3). Not yet updated.



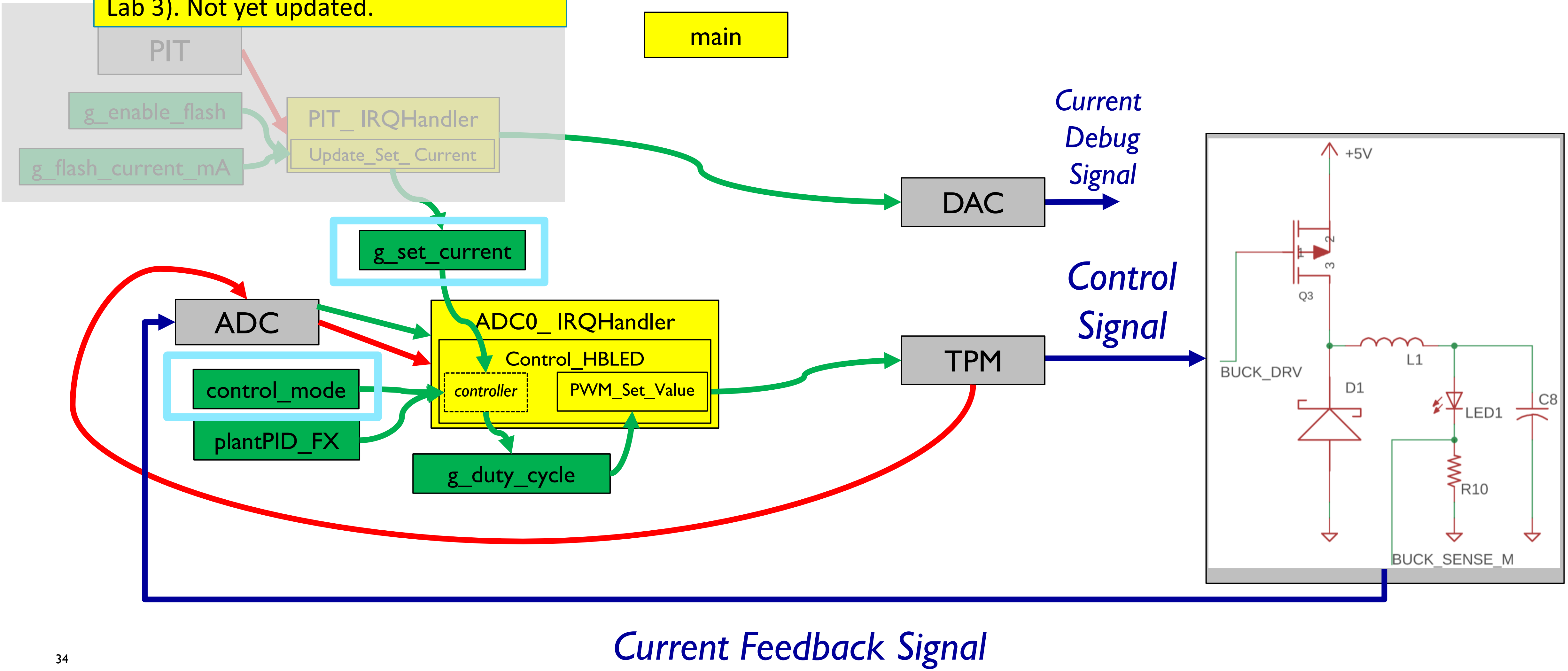
Lab 3: Q. 6-7 (Asynch. Sampling, Closed-Loop, No Transients)

Note:
- These slides refer to questions in an old version of Practical HW3 (was called Lab 3). Not yet updated.



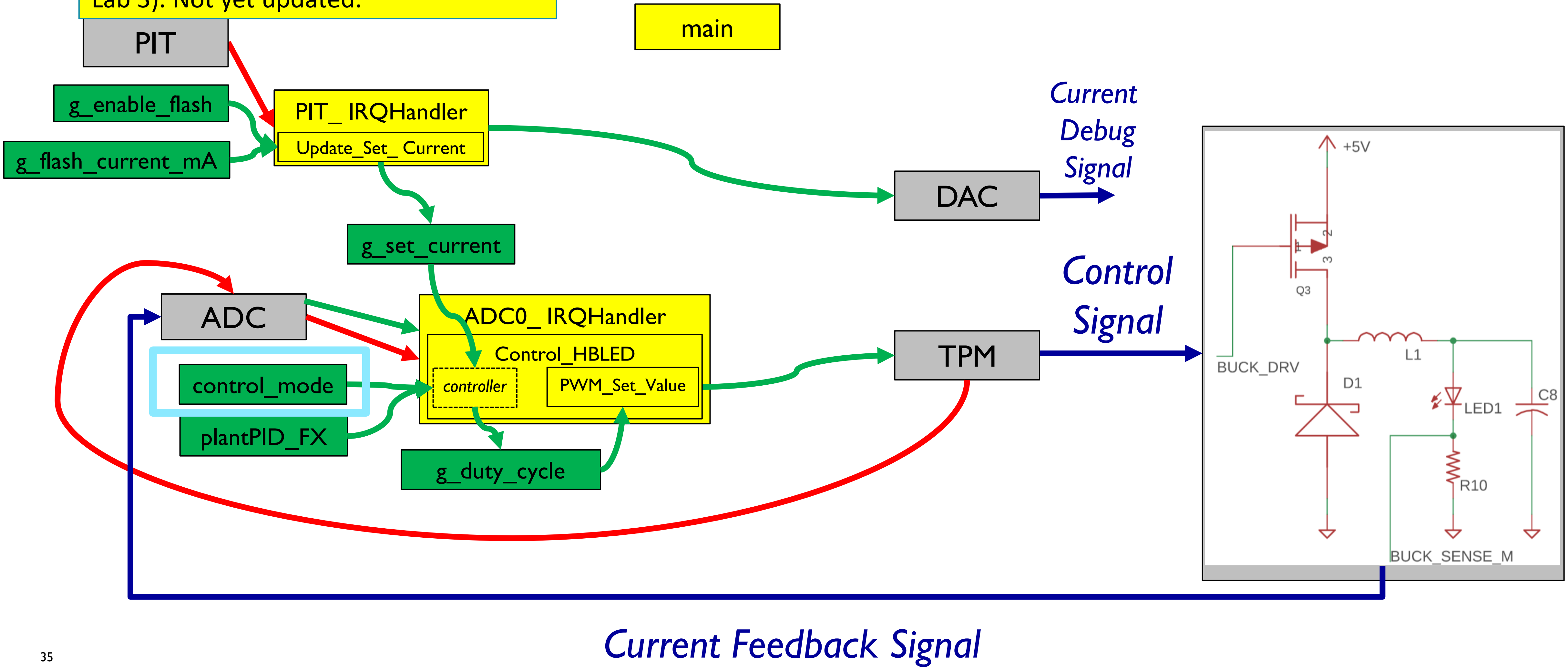
Lab 3: Q.8-10,13 (Synch. Sampling, Closed-Loop, No Tr

Note:
- These slides refer to questions in an old version of Practical HW3 (was called Lab 3). Not yet updated.



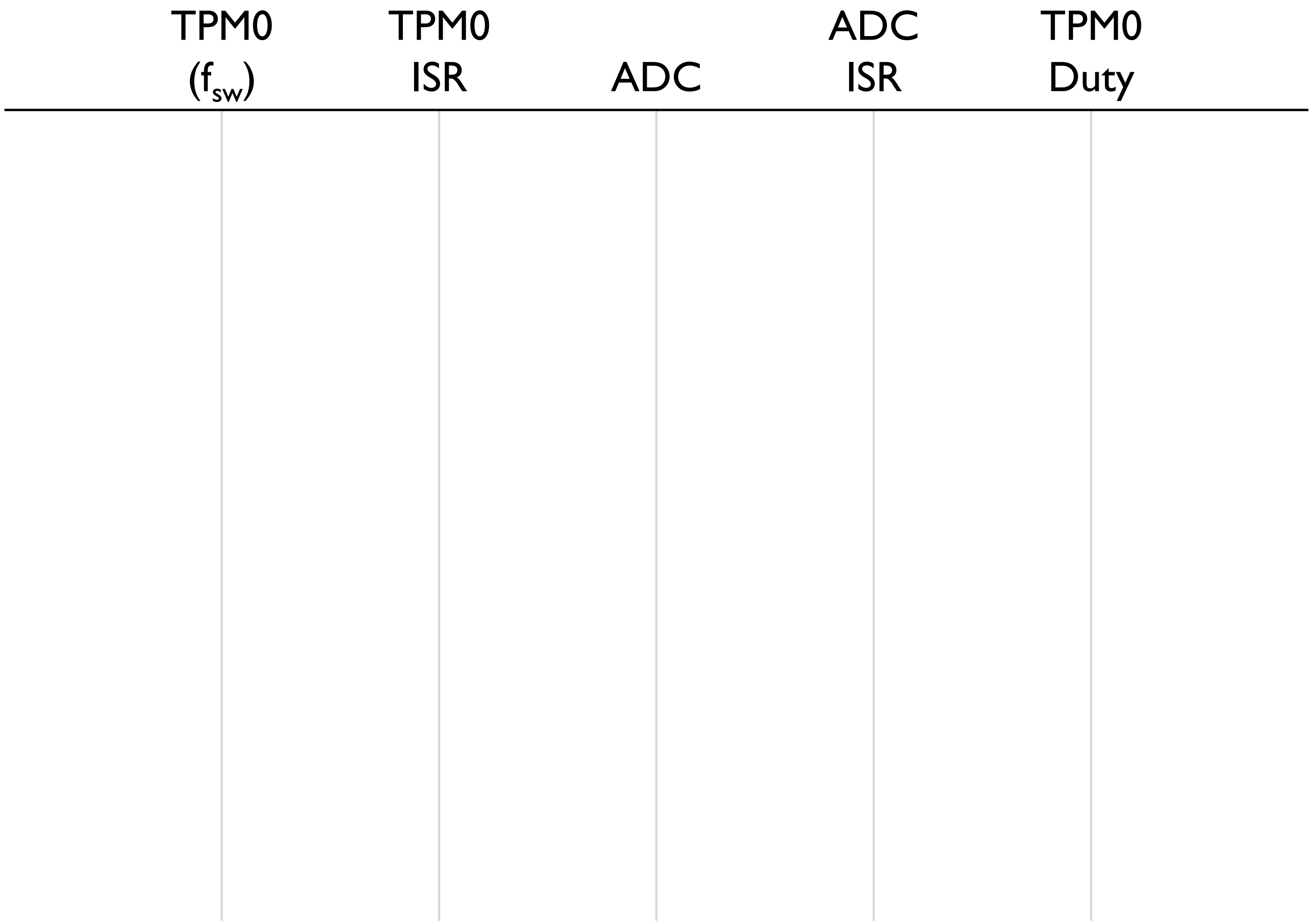
Lab 3: Q.11-12, 14-15 (Synch. Sampling, Closed-Loop, Tr

Note:
- These slides refer to questions in an old version of Practical HW3 (was called Lab 3). Not yet updated.



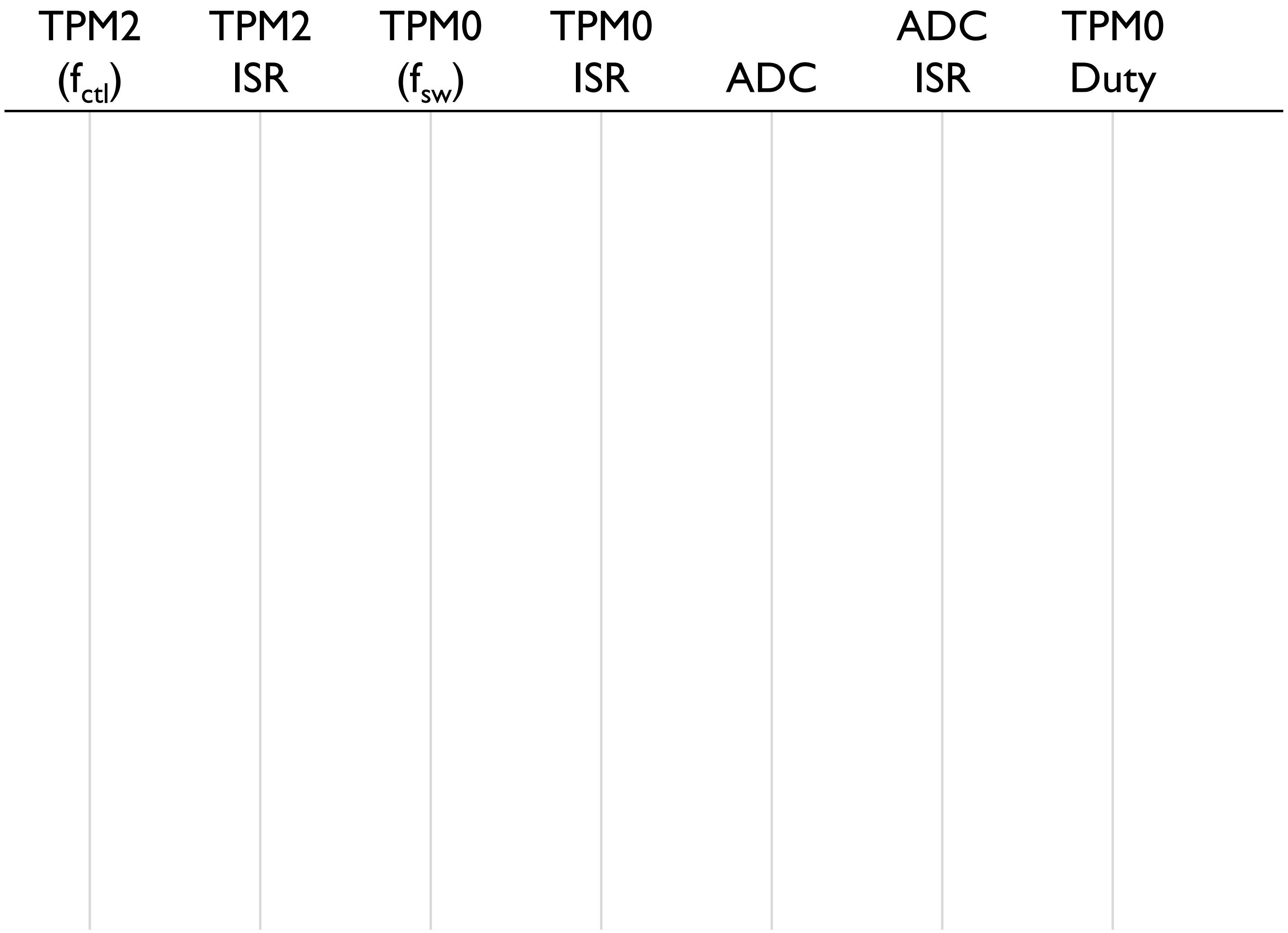
SYNC Sequence Diagram, Software Division for F_{ctl}

USE_SYNC_SW_CTL_FREQ_DIV



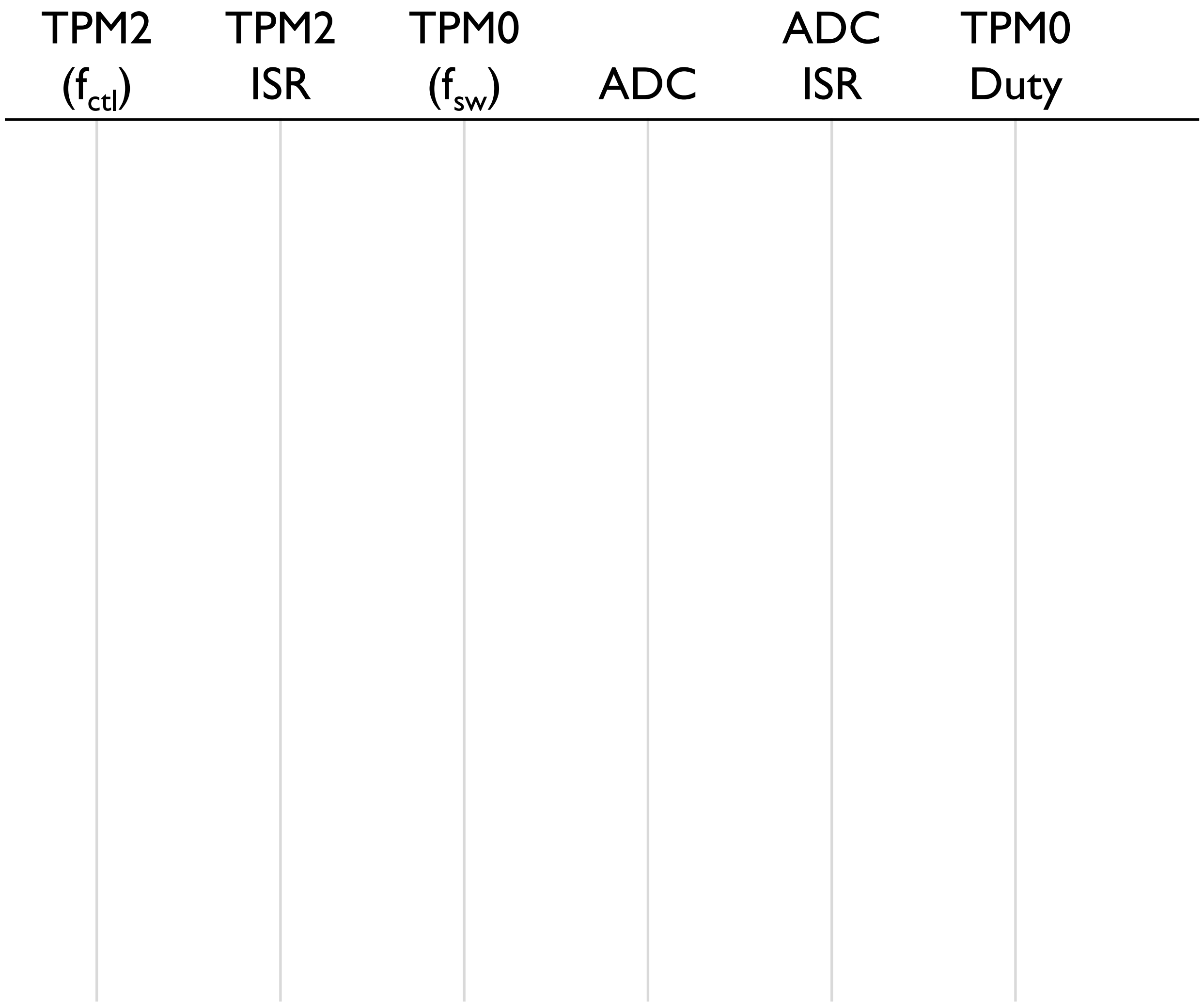
SYNC Sequence Diagram, 2nd Timer for F_{ctl}, ADC SW Trigger

(not in your code)



SYNC Sequence Diagram, 2nd Timer for F_{ctl}, ADC HW Trigger

`(not in your code)`



How About Using DMA?

(not in your code)

- Replace TPM2 ISR with DMA transfer to enable ADC hardware trigger

TPM2 (f_{ctl})	DMA	TPM0 (f_{sw})	ADC	ADC ISR	TPM0 Duty

Old Diagrams and Notes

Why Use Switch-Mode? Efficiency

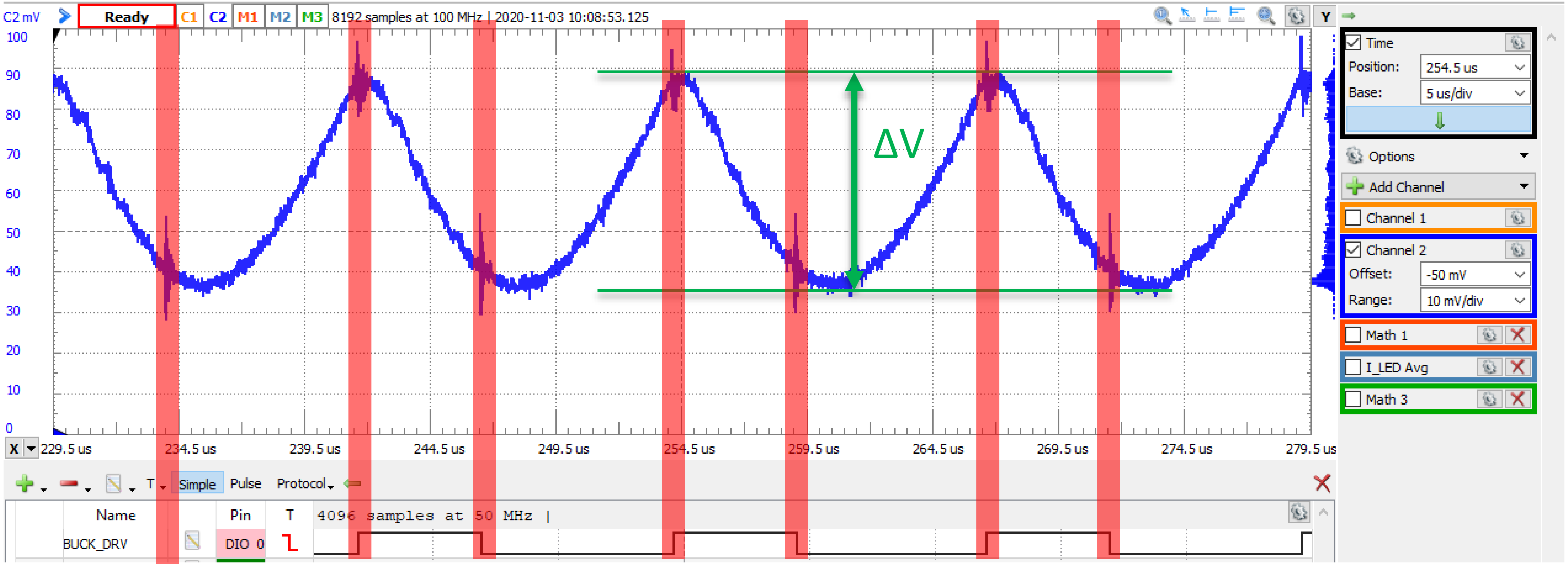
- Linear regulator
  - Drops input voltage using “pass” transistor as variable resistor, converting extra power to heat.
  - If output doesn’t need as much power, reduce conductance of transistor (raise resistance), wasting much more power.
- Switch-mode regulator
  - Drop voltage by connecting and disconnecting input voltage very frequently (faster than load circuit can notice) to create lower average output voltage.
  - If output doesn’t need as much power, reduce duty cycle (fraction of time) input is connected. Minor additional power loss.
    - Switches (transistors, diodes) are either on (very low resistance) or off (open circuit)
    - Duty cycle typically controlled by pulse-width modulated (PWM) signal
  - If needed, filter output voltage to smooth it

NC STATE UNIVERSITY

Energy Storage and Filtering

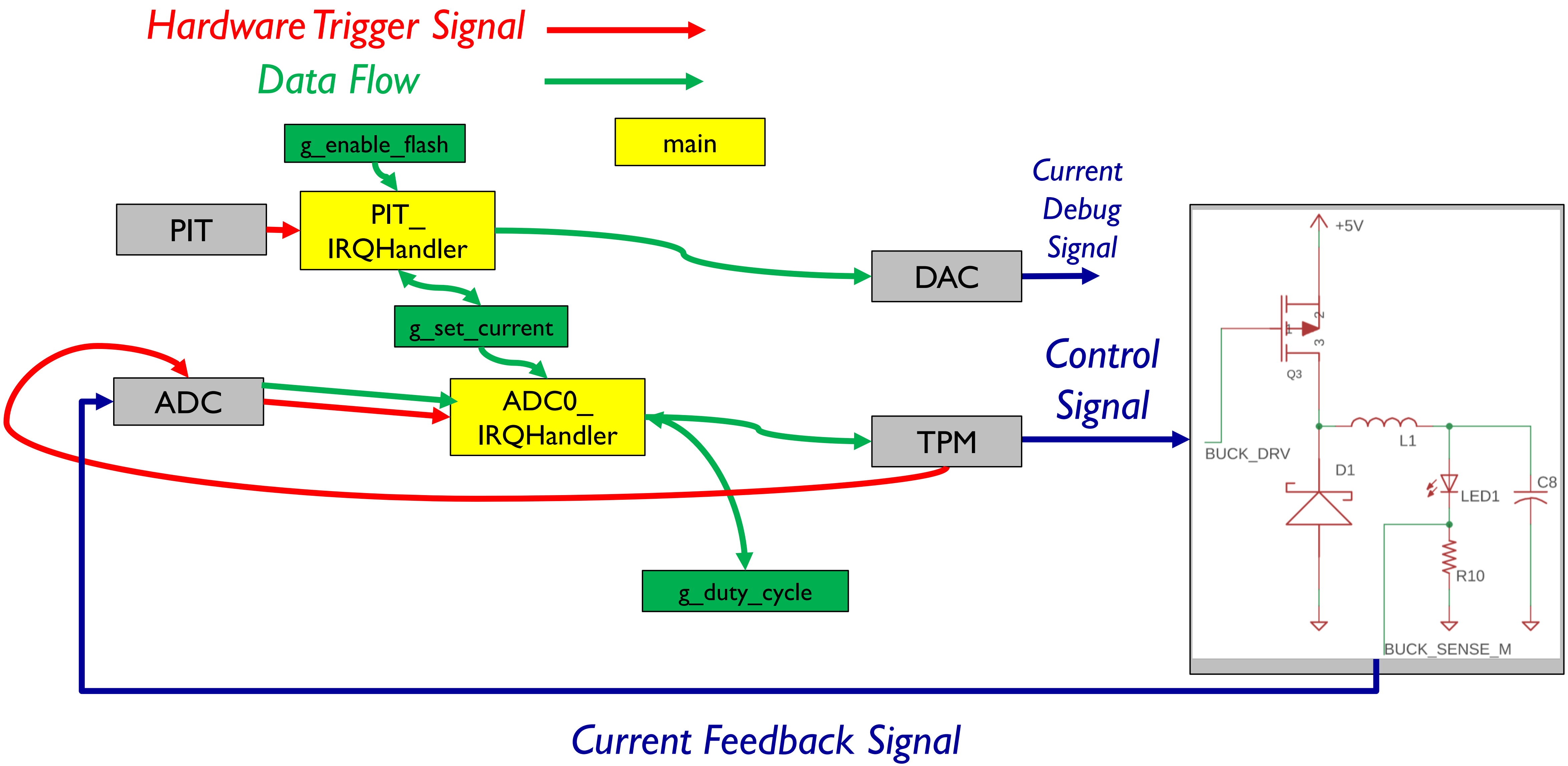
- Use capacitors and inductors to store energy and filter out ripple
  - Capacitors – can't instantaneously change voltage
  - Inductors – can't instantaneously change current
- Power converted between electrical and magnetic forms for efficient storage and filtering
- Capacitor stores energy as voltage difference
  - $E = ½ CV^2$
  - *Total storage limited by voltage: need higher voltage*
  - *Current affects charge, discharge rate, but not total energy storage*
- Inductor stores energy as current and magnetic field
  - $E = ½ LI^2$
  - *Total storage limited by current: need higher current*
  - *Voltage affects charge, discharge rate, but not total energy storage*
  - *Good match for low voltage operation. Easier to increase current capacity.*

When to Sample the LED Current?

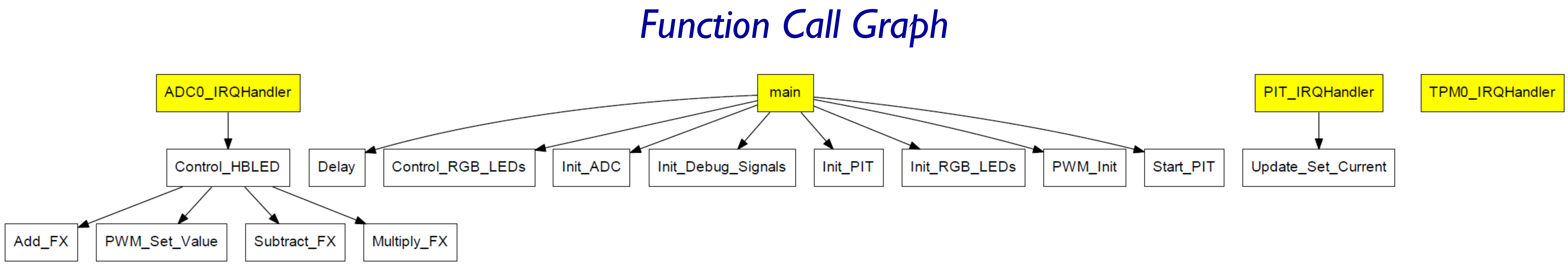


- Need to sample in phase with BUCK_DRV signal (PWM)
  - Same phase for each sample, otherwise may have error of up to ΔV in each sample
  - Trigger with TPM
- Stay away from times of BUCK_DRV transitions, since they are noisy

Hardware and Software Overview

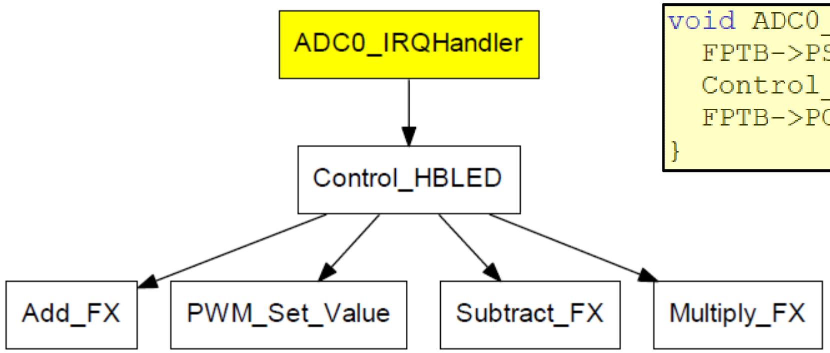


Which Functions Can Call Each Other?



- Functions in yellow are threads or interrupt handlers

Examining ADC0_IRQ

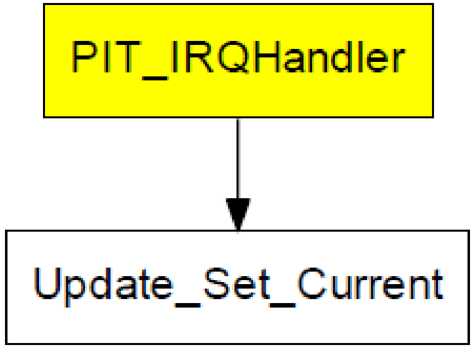



```
void ADC0_IRQHand
  FPTB->PSOR = MA
  Control_HBLED()
  FPTB->PCOR = MA
}
```

- Global variable accesses
  - Reads g_set_current
  - Reads and writes g_duty_cycle
- Peripheral access
  - Reads ADC0 result register
  - Writes g_duty_cycle to TPM0

```
    ; // wait until end of conversion
#endif
  res = ADC0->R[0];

  measured_current = (res*1500)>>16; // Extra Credit: Make this code work: V_REF_MV*MA_SCALING_F

  switch (control_mode) {
    case OpenLoop:
        // don't do anything!
      break;
    case BangBang:
      if (measured_current < g_set_current)
        g_duty_cycle = LIM_DUTY_CYCLE;
      else
        g_duty_cycle = 0;
      break;
    case Incremental:
      if (measured_current < g_set_current)
        g_duty_cycle += INC_STEP;
      else
        g_duty_cycle -= INC_STEP;
      break;
    case Proportional:
      g_duty_cycle += (pGain_8*(g_set_current - measured_current))/256; //  - 1;
    break;
    case PID:
      g_duty_cycle += UpdatePID(&plantPID, g_set_current - measured_current, measured_current);
      break;
    case PID_FX:
      error_FX = INT_TO_FX(g_set_current - measured_current);
      change_FX = UpdatePID_FX(&plantPID_FX, error_FX, INT_TO_FX(measured_current));
      g_duty_cycle += FX_TO_INT(change_FX);
    break;
    default:
      break;
  }

  // Update PWM controller with duty cycle
  if (g_duty_cycle < 0)
    g_duty_cycle = 0;
  else if (g_duty_cycle > LIM_DUTY_CYCLE)
    g_duty_cycle = LIM_DUTY_CYCLE;
  PWM_Set_Value(TPM0, PWM_HBLED_CHANNEL, g_duty_cycle);
  FPTB->PCOR = MASK(DBG_CONTROLLER);
}
```

Examining PIT_IRQHandler



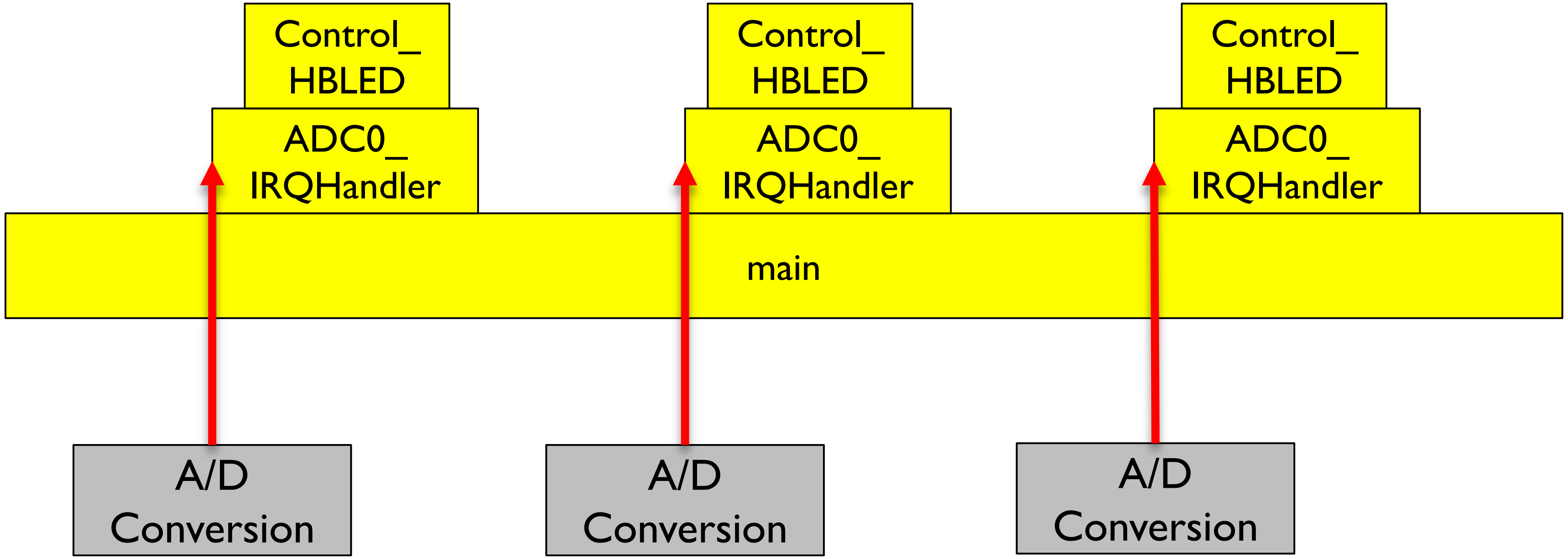

```c
void PIT_IRQHandler() {
  // check to see which channel triggered interrupt
  if (PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
    // clear status flag for timer channel 0
    PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
    // Do ISR work
    Update_Set_Current();
  } else if (PIT->CHANNEL[1].TFLG & PIT_TFLG_TIF_MASK) {
    // clear status flag for timer channel 1
    PIT->CHANNEL[1].TFLG &= PIT_TFLG_TIF_MASK;
  }
}
```

```c
void Update_Set_Current(void) {
  static int delay=FLASH_PERIOD;

  if (g_enable_flash){
    delay--;
    if (delay==1) {
      FPTB->PSOR = MASK(DBG_LED_ON);
      g_set_current = FLASH_CURRENT_MA;
      Set_DAC_mA(g_set_current);
    } else if (delay==0) {
      delay=FLASH_PERIOD;
      FPTB->PSOR = MASK(DBG_LED_ON);
      g_set_current = FLASH_CURRENT_MA/4;
      Set_DAC_mA(g_set_current);
    } else {
      FPTB->PCOR = MASK(DBG_LED_ON);
      g_set_current = 0;
      Set_DAC_mA(g_set_current);
    }
  }
}
```

```c
void Set_DAC_mA(unsigned int current) {
  unsigned int code = MA_TO_DAC_CODE(current);
  // Force 16-bit write to DAC
  uint16_t * dac0dat = (uint16_t *)&(DAC0->DAT[0].DATL);
  *dac0dat = (uint16_t) code;
}
```

- Global variable accesses
  - Reads g_enable_flash
  - Reads and writes g_set_current
- Peripheral access
  - Writes to the DAC
- *Note: compiler inlined call to Set_DAC_mA*

Rough Timing Diagram



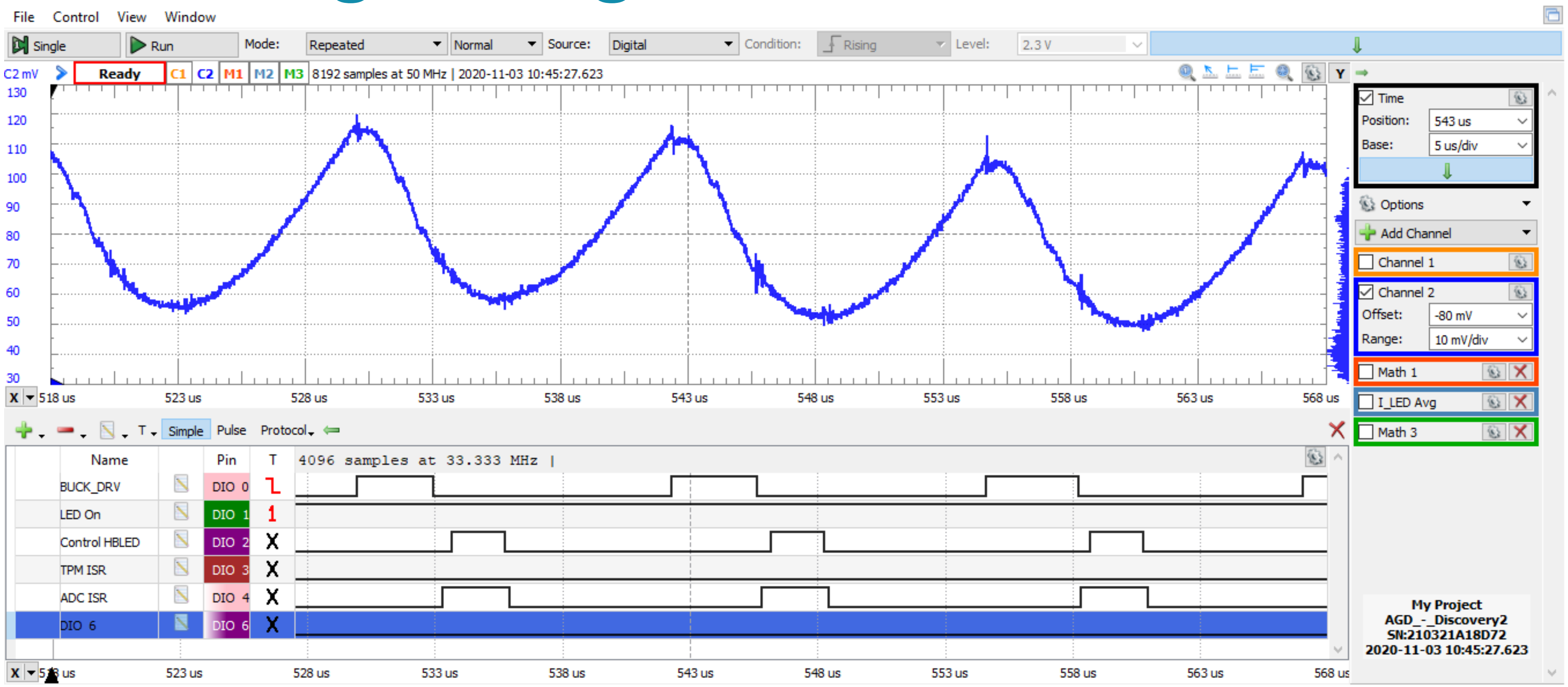
- Does it really look like this?

NC STATE UNIVERSITY

Examining Timing with Instrumentation

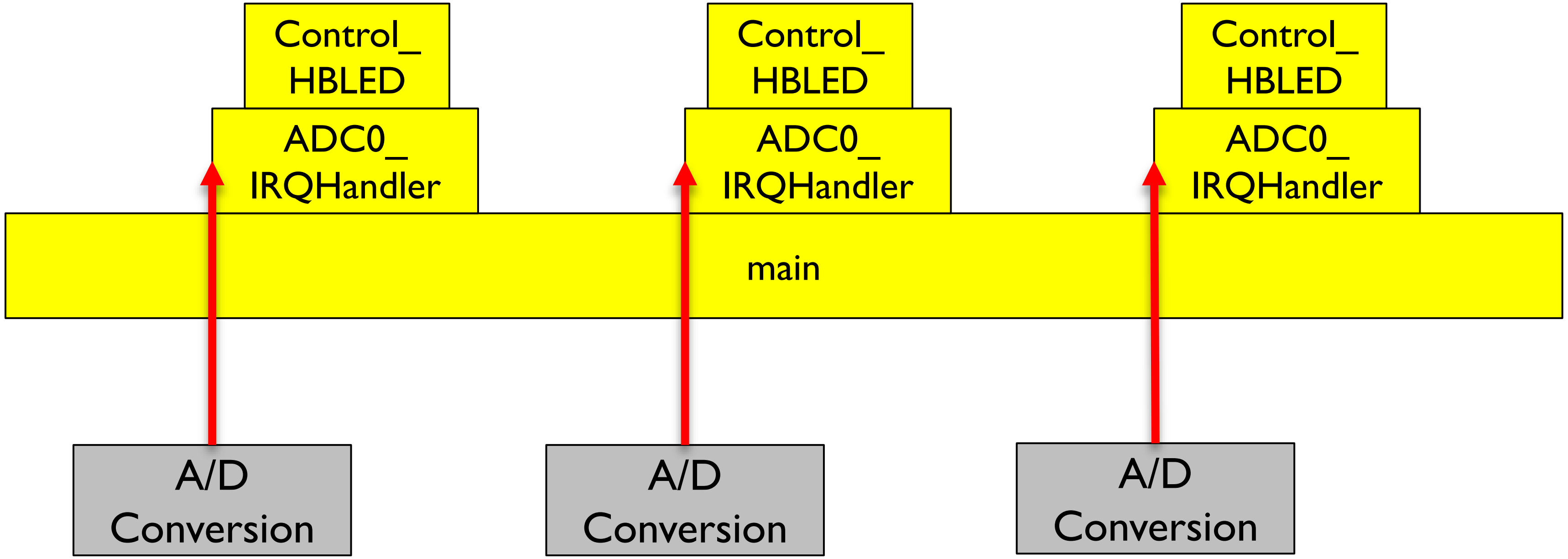
- How long does it take to do the whole control loop?
- How long does each individual part take?
- How stable is the timing?
- When do we really sample the input voltage?

Examining Timing with Instrumentation



- Hardware signals
  - BUCK_DRV: when does cycle start, end
  - Current feedback: see where the noise is
- Software-generated signals
  - ADC ISR: when does AD conversion finish, ISR start
  - Control_HBLED: when control loop runs
  - LED On: helps with filtering triggers
- Would right control mode let us see signals better?
- How to trigger?
  - Using center-aligned PWM, so not ideal to trigger off rising or falling edge of BUCK_DRV
- PIT ISR interference with ADC ISR

Rough Timing Diagram



- Does it really look like this?

Old: low-true

New: high-true