New issue

Jump to bottom

Setting the trigger #100

Open **benalcazardiego** opened this issue on 17 Feb 2017 · 10 comments

benalcazardiego commented on 17 Feb 2017

Hello. I have a doubt. I do not really know what is the trigger and how should I use it? I need to adquire data to a buffer and save it to a file only when the input in a channel is bigger than a specific value.

jeras commented on 17 Feb 2017 Contributor

The basic steps are:

1. set decimation
2. set trigger level
3. for a noisy signal set trigger hysteresis
4. set post trigger delay (in API value 0 means half buffer in the experimental API1 this is just the number of samples stored into the buffer after trigger)
5. start acquisition
6. wait a bit for the buffer to load some data before trigger
7. set the trigger source
8. wait for trigger source register to be cleared, this indicates, writing into the buffer has ended
9. read the data

This is code from a Jupyter example, so API1 and Python, but C functions have similar names:

```
rp.AcqSetDecimationFactor(1)
rp.AcqSetTriggerLevel(0, 0.1)
rp.AcqSetPostTriggerDelay(size//2) # place trigger in the middle of the buffer

# start acquisition process
rp.AcqStart()

# Here you should wait a bit since pre trigger settings are not yet implemented in FPGA.

# set trigger source to start acquisition
rp.AcqSetTriggerSrc(rp.TRIG_SRC_CHA_PE)
# wait in a loop for trigger state to chage from TRIG_STATE_WAITING
while rp.AcqGetTriggerSrc():
    pass

# read data from FPGA FIFO into memory and display it
buff = [rp.AcqGetOldestDataV(ch, size) for ch in channels];
```

If you are using the original non experimental API, you the functions behave differently. Most examples are probably wrong regarding the function used to check trigger state, `AcqGetTriggerSrc` should be used. This is due to legacy FPGA code.

jeras commented on 17 Feb 2017 Contributor

API1 is not stable and will probably not be for some time, since I plan to also update the FPGA.

benalcazardiego commented on 18 Feb 2017 Author

I have some questions I hope you can help me with.

1. I do not get one thing about the trigger. What I understand is that no data will be saved in the ADC buffer until the input value in one of the frontend channel gets to the defined value. If that is the case why does it say in the example "acquire_trigger_posedge.c" that after the "rp_AcqStart();" command one should wait a moment with a sleep ("After acquisition is started some time delay is needed in order to acquire fresh samples in to buffer") while we have not asked for the trigger_state yet so we do not know if the trigger value has been reached? It says that we shall wait for the buffer to fill but we do not know if the trigger has been reached. Now you also said that one has to "wait a bit for the buffer to load some data before trigger" and once again I though the buffer will not load any data until the trigger was reached so I do not get well how the trigger and the ADC buffer work.
2. In the moment the trigger level is reached, I understood that a number of samples defined by rp_AcqSetTriggerDelay(trigger_delay) is saved to the ADC buffer. Do you mean that 0 is a predefined value and that when you use it half of the 16k of the ADC buffer will be filled with data? Also, in the table that describes this example in http://redpitaya.readthedocs.io/en/latest/doc/appsFeatures/examples/acqRF-exm1.html
   Sampling Rate Time scale/length of a buffer Trigger delay in samples Trigger delay in seconds
   125 MS/s 131.072 us from - 8192 to x -6.554E-5 to x
   what is the meaning of x, what does "from - 8192 to x" mean, where does 8192 come from? This table refers to the case when the full buffer (16k) is used, right? because I thought that the length of the buffer used depended on the trigger_delay parameter. Also, what does it mean that "place trigger in the middle of the buffer"
3. In the same example "acquire_trigger_posedge" right before starting the acquisition the state is set to triggered ("rp_acq_trig_state_t state = RP_TRIG_STATE_TRIGGERED;"). Should not it be WAITING so the program wait until a trigger is reached? How does the state works? I thought that it works so that after the acquisition starts (rp_AcqStart();) a trigger is expected. When it is reached, the samples defined by trigger_delay are saved to the ADC buffer during a time that depends on the samples rate (defined by sample rate or decimation). When the last sample has been saved, the state automatically changes to waiting or does it stay as triggered until I change it manually? After the trigger is triggered a sleep is executed. That sleep is used so that the buffer fills with the samples just after the trigger has been reached, right? or the state TRIGGERED is set after the data is saved to the buffer?
4. Can I define the source before the start?
5. When the sleep after the state TRIGGERED is reached is executed, should it be exactly equal to the time the buffer takes to get full with the specified samples? If it is too big, is there the possibility that if it too big the ADC buffer gets new data? This question only has sense asuming that the state changes automatically from triggered to waiting and as i mentioned earlier I am not sure about that.

Thanks in advance for your help. I am learning all about the red pitaya by myself and the information in the internet is not that clear so I appreciate a lot your help.

benalcazardiego commented on 18 Feb 2017 • edited Author

otherwise I would not consider it.

I made this program. I used a bigger width pulse as an example (6 us).

```
float trigger_level = 0.4; /Only pulses with values higher than 0.4 will be saved/
float measure_time = 10; /*The acquisition will last 10 seconds */
FILE *fd;
fd = fopen("Filename.dat", "wb");
uint32_t buff_size = 750; /*Since the pulse width is 6 us, then 750 samples with 125MSps (decimation 1) will take the entire pulse width */
float *buff = (float *)malloc(buff_size * sizeof(float));
if(rp_Init() != RP_OK)
{
fprintf(stderr, "\tError: Rp api init failed!\n");
return EXIT_FAILURE;
}
rp_acq_decimation_t acq_decimation = RP_DEC_1;
float ADC_trigger_level = 0.1; /This way the pulse is got from the beginning before it reaches 0.4/
int32_t trigger_delay = (int32_t)buff_size;
rp_AcqReset();
rp_AcqSetDecimation(acq_decimation);
rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE);
rp_AcqSetTriggerLevel(ADC_trigger_level);
rp_AcqSetTriggerDelay(trigger_delay);
rp_acq_trig_state_t state = RP_TRIG_STATE_TRIGGERED; /Do not know if it should be waiting or triggered /
float impulse_period = 6pow(10,-6);
struct timespec tim;
tim.tv_nsec = impulse_periodpow(10,9);
clock_t start, end;
volatile double elapsed;
start = clock();
while(1)
{
end = clock();
elapsed = ((double) (end-start)) / (double) CLOCKS_PER_SEC;
if(elapsed >= measure_time)
{
break;
}
rp_AcqGetTriggerState(&state);
if(state == RP_TRIG_STATE_TRIGGERED)
{
nanosleep(&tim, NULL); /%This is used so the ADC_buffer is filled with the impulse data
rp_AcqGetOldestDataV(RP_CH_2, &buff_size, buff
bool trigger_reached = 0;
for(i = 0; i < buff_size; i++)
{
if (buff[i] >= trigger_level)
{
trigger_reached = 1; /%If there is a value bigger than 0.4 the buffer is saved%/
break;
}
}
if (trigger_reached == 1)
{
for(i = 0; i < buff_size; i++)
{
fprintf(fd,"%f\n", buff[i]);
}
}
}
}
fclose(fd);
rp_Release()
```

Everything works fine except the acquisition. It stops working in the moment if finds a trigger_ADC (in this case in the moment it reaches 0.1 V), so there may be a problem with the nanosleep. When I use usleep it does not stop working but it seems that the specified time after getting the trigger triggered is too long so the ADC buffer fills with new data and the pulse is lost. Then no data will be saved to the file. I do not know if this is the problem in fact or I am not understanding something right.
Thanks for your help and time. In practice, I need to get pulses with shorter width (300 ns) so it is more difficult

HrRossi commented on 18 Feb 2017 • edited — Contributor

> What I understand is that no data will be saved in the ADC buffer until the input value in one of the frontend channel gets to the defined value.

You misunderstood. As soon as you arm the oscilloscope (eg. by calling rp_AcqStart), writing into the buffer starts. When the 16k ringbuffer has been filled once, the oldest data in it will be overwritten, so the buffer will always contain the latest 16k samples.
When a trigger event is recognized, a countdown starts from (TriggerDelay + 8192). When the counter reaches 0, recording stops and the TriggerSrc is set to IDLE.

> What I want to do exactly is a program which works for a specific time, during that time some pulses with a really short width (300 ns) have to be collected and none can be lost. I need the complete pulse.

Let's say the level above which you want to save the pulse is `Vsave`, your maximum expected pulse width is `Lmax`, and the maximum expected time from 0 to `Vsave` is `Lrampup`.

1. Set TriggerLevel to `Vsave` and the TriggerDelay to `Lmax`.
2. Call AcqStart and wait for `Lrampup`.
3. Set the TriggerSrc to RP_TRIG_SRC_CHB_PE and wait for it to reset to IDLE - this happens when the TriggerDelay countdown is finished after a trigger.
4. Read `Lmax + Lrampup` samples with AcqGetLatestData... and save them.
5. Repeat from 3.

benalcazardiego commented on 20 Feb 2017 — Author

Oh I see now. That clarifies things a lot. Thank you ver much. Would there be any problem with the following sequence then? I am not sure about defining the trigger state variable in waiting or in triggered like in the example and if to define it in that place.

```
rp_AcqSetDecimation(RP_DEC_1;);
rp_AcqSetTriggerLevel(0.1);
rp_AcqSetTriggerDelay((int32_t)750); /750 samples for 6 us/
rp_acq_trig_state_t state = RP_TRIG_STATE_WAITING;
clock_t start, end;
volatile double elapsed;
float trigger_level = 0.4;
start = clock();
rp_AcqStart();
usleep(135); /135 us for 16386 samples and to fill the 16k ADC buffer/
while(1)
{
end = clock();
elapsed = ((double) (end-start)) / (double) CLOCKS_PER_SEC;
//fprintf(stdout,"Elapsed time:%f\n",elapsed);
if(elapsed >= measure_time)
{
break;
}
rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE;
rp_AcqGetTriggerState(&state);
if(state == RP_TRIG_STATE_TRIGGERED)
{
usleep(75); /*75 us for 8192 + 750 samples */
rp_AcqGetOldestDataV(RP_CH_2, &buff_size, buff);
}
bool trigger_reached = 0;
for(i = 0; i < buff_size; i++)
{
if (buff[i] >= trigger_level)
{
trigger_reached = 1;
break;
}
}
if (trigger_reached == 1)
{
for(i = 0; i < buff_size; i++)
{
fprintf(fd,"%f\n", buff[i]);
}
}
}
}
```

jeras commented on 20 Feb 2017 — Contributor

Please use markdown for the code, without proper indents it is difficult to read. So I am not sure if your while loop is intended for pooling the trigger status.

The trigger state function is rather useless, lets say it is there for backward compatibility, but I am not sure.
Instead rp_AcqGetTriggerSrc should be used, a red value of 0 indicates trigger was detected and all post trigger data was loaded.
The order should be something like:

```
rp_AcqStart();
usleep(135); /135 us for 16386 samples and to fill the 16k ADC buffer/
rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE;
while(rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE);
// no need to wait here
rp_AcqGetOldestDataV(RP_CH_2, &buff_size, buff);
```

benalcazardiego commented on 20 Feb 2017 • edited — Author

Is there anyway to avoid the countdown to be 8192+trigger_delay but only trigger_delay?

benalcazardiego commented on 21 Feb 2017 — Author

In the rp.h it says that
"/**

- Sets the trigger source used at acquiring signal. When acquiring is started,
- the FPGA waits for the trigger condition on the specified source and when the condition is met, it
- starts writing the signal to the buffer.
- **@param** source Trigger source.
- **@return** If the function is successful, the return value is RP_OK.
- If the function is unsuccessful, the return value is any of RP_E* values that indicate an error.

*/
int rp_AcqSetTriggerSrc(rp_acq_trig_src_t source);
"
Is that wrong then?

benalcazardiego commented on 21 Feb 2017 — Author

Oh sorry for the indentation problem. He is the last code I have written for a pulse width of 6us. Before starting acquiring data, the program sets an output level gradually to a specific value. The inputs are entered to the program.

```
#include "math.h"
#include <inttypes.h>

#define VERSION "0.1"

int main (int argc, char **argv) {


  /* Syntax analysis */
  if (argc != 5)
  {
          fprintf(stderr,"\tError: Incorrect syntax.\n Syntax:  LD_LIBRARY_PATH=/opt/redpitaya/lib ./%s [Filename] [Trigger Level] [Measure time] [HV 0] \nExample:
  LD_LIBRARY_PATH=/opt/redpitaya/lib ./%s data.dat -0.2 10 1.2 \n",argv[0],argv[0]);
          return EXIT_FAILURE;
  }

  if (atof(argv[4]) < 0 || atof(argv[4]) > 2.5)
  {
          fprintf(stderr,"\tError: Incorrect input value.\n HV must satisfy the condition 0 [V] <= HV <= 2.5 [V]");
          return EXIT_FAILURE;
  }

  /* Definition of the input parameters */
  float trigger_level = atof(argv[2]);
  float measure_time = atof(argv[3]);
  float HV = atof(argv[4]);



  /* General information */
  fprintf(stdout,"\t%s version %s\n",argv[0],VERSION);
  fprintf(stdout,"\tThe HV value sent is %f (V)\n",1.4*HV);

  /* Open file and test for error */
  FILE *fd;
  fd = fopen(argv[1], "wb");
  if (fd == NULL)
  {
          fprintf(stderr, "\tError: Unable to input file %s \n", argv[1]);
          return EXIT_FAILURE;
  }

  /* Create buffer */
  uint32_t buff_size = 8942; /*8192 + 750*/
  float *buff = (float *)malloc(buff_size * sizeof(float));

  /* Initialization of the RP */
  if(rp_Init() != RP_OK)
  {
      fprintf(stderr, "\tError: Rp api init failed!\n");
      return EXIT_FAILURE;
  }

  /* HV setting */
  rp_AOpinReset();
  float step = 0.001;
  float base = 0;
  float tiempo = 5;

  /* Definition of the base value (may be ommited) */
  if(rp_AOpinGetValue(0, &base) != RP_OK)
  {
      fprintf(stderr, "\tError: HV base value establishment failed\n");
      return EXIT_FAILURE;
  }

  /* 0 to HV setting */
  sleep(3);
  float HV_points =  (HV-base)/step;
  float period = tiempo/HV_points;
  int i = 0;
  float ramp = base;
  fprintf(stderr, "\t0 to HV value establishment started\n");
  for (i=0; i<HV_points; i++)
  {
          ramp = ramp + step;
          if (rp_AOpinSetValue(0, ramp) != RP_OK)
          {
                  fprintf(stderr, "\tError: 0 to HV value establishment failed\n");
          return EXIT_FAILURE;
          }
          usleep(period*pow(10.0,6));
  }

  /* HV corrovoration */
  float HV_value = 0;
  if(rp_AOpinGetValue(0, &HV_value) != RP_OK)
  {
      fprintf(stderr, "\tError: HV value corrovoration failed\n");
      return EXIT_FAILURE;
  }
  fprintf(stdout, "\tEstablished HV value: %f\n", HV_value);


  /*** Start of the data adquisition process ***/

  fprintf(stdout, "\tTo start the data adquisition process, press ENTER");
  getchar();

  /* Definition of the acquire parameters */
  rp_acq_decimation_t acq_decimation = RP_DEC_1;
  float ADC_trigger_level = 0.31;
  int32_t trigger_delay = (int32_t)buff_size;

  /* Reset of the acquire writing state machine */
  if(rp_AcqReset() != RP_OK)
  {
      fprintf(stderr, "\tError: Reset of the acquire writing state machine failed\n");
      return EXIT_FAILURE;
  }

  /* Set of the adquisition decimation */
  if(rp_AcqSetDecimation(acq_decimation) != RP_OK)
  {
      fprintf(stderr, "\tError: Set of the adquisition decimation failed\n");
      return EXIT_FAILURE;
  }

  /* Set of the acquisition trigger value */
  if(rp_AcqSetTriggerLevel(ADC_trigger_level) != RP_OK)
  {
      fprintf(stderr, "\tError: Set of the adquisition trigger value failed\n");
```

```
/* Set of the ADC buffer length */
if(rp_AcqSetTriggerDelay(trigger_delay) != RP_OK)
{
    fprintf(stderr, "\tError: Set of the adquisition trigger length failed\n");
    return EXIT_FAILURE;
}

//struct timespec tim;
//tim.tv_nsec = impulse_period*pow(10,9);

/* Start of the adquisition loop*/
clock_t start, end;
volatile double elapsed;
start = clock();

if(rp_AcqStart() != RP_OK)
{
        fprintf(stderr, "\tError: Start of the Acquisition failed\n");
        return EXIT_FAILURE;
}

usleep(135);

while(1)
{
        /* Set of the adquisition trigger source */
        if(rp_AcqSetTriggerSrc(RP_TRIG_SRC_CHB_PE) != RP_OK)
        {
        fprintf(stderr, "\tError: Set of the adquisition trigger source failed\n");
        return EXIT_FAILURE;
        }

        rp_acq_trig_state_t state = RP_TRIG_STATE_WAITING;

    while(1)
        {
                end = clock();
        elapsed = ((double) (end-start)) / (double) CLOCKS_PER_SEC;
        //fprintf(stdout,"Elapsed time:%f\n",elapsed);
                if(elapsed >= measure_time)
        {
                goto timeover;
        }

        rp_AcqGetTriggerState(&state);

        if(state == RP_TRIG_STATE_TRIGGERED)
                {
                usleep(75);
                break;
        }
        }

        if(rp_AcqGetOldestDataV(RP_CH_2, &buff_size, buff) != RP_OK)
        {
        fprintf(stderr, "\tError: Buffer filling failed\n");
        return EXIT_FAILURE;
    }

    bool trigger_reached = 0;

        for(i = 0; i < buff_size; i++)
        {
                if (buff[i] >= trigger_level)
                {
                trigger_reached = 1;
                break;
                }
        }

        if (trigger_reached == 1)
        {
                for(i = 0; i < buff_size; i++)
                {
                        /*
                        end = clock();
                elapsed = ((double) (end-start)) / (double) CLOCKS_PER_SEC;
                */
                        int half_diff=375;
                        int sup_limit=8942;
                        int inf_limit=0;

                        if (i-half_diff>inf_limit)
                        {
                                inf_limit=i-half_diff;
                        }

                        if (sup_limit-i>half_diff)
                        {
                                sup_limit=i+half_diff;
                        }

                        int j=0;

                        for (j=inf_limit;j<sup_limit;j++)
                        {
                                if (buff[j]>=trigger_level)
                                {
                                        //fprintf(fd,"%lf\t%f\n", elapsed, buff[i]);
                                        fprintf(fd,"%f\n", buff[i]);
                                        break;
                                }
                        }
                }
        }
}

timeover:
fprintf(stdout, "\tThe data adquisition process is over\n");
/*** End of the data adquisition process ***/

/* HV to 0 setting */
sleep(3);
i = 0;
ramp = HV_value;
fprintf(stderr, "\tHV to 0 value establishement started\n");

for (i=0; i<HV_points; i++)
{
        ramp = ramp - step;

        if (rp_AOpinSetValue(0, ramp) != RP_OK)
        {
```

```
        usleep(period*pow(10.0,6));
    }

    /* Final HV corrovoration */
    float HV_final_value = 0;

    if(rp_AOpinGetValue(0, &HV_final_value) != RP_OK)
    {
        fprintf(stderr, "\tError: final HV value corrovoration failed\n");
        return EXIT_FAILURE;
    }

    fprintf(stdout, "\tEstablished final HV value: %f \n", HV_final_value);

    /* Close of the file */
    fclose(fd);

    /* Reset of the RP
    if(rp_Reset() != RP_OK)
    {
        fprintf(stderr, "Error: Rp api res failed!\n");
        return EXIT_FAILURE;
    }
    */

    /* Release of the RP */
    if(rp_Release() != RP_OK)
    {
        fprintf(stderr, "\tError: Rp api release failed!\n");
        return EXIT_FAILURE;
    }

    free(buff);
    fprintf(stdout, "\tThe application has finished working properly\n");
    return EXIT_SUCCESS;


}
```

What do you guys think? I have proved it and it seems to be working correctly except for one thing. When I start the app, and again when I press enter to start the data acquisition process, I see that the oscilloscope app from the RP kind of freezes except for a little fragment at the start of the time axis. In this case the pulse collected is really low (1 when working for 100 seconds). However, this problem is solved when I press autoscale in the RP oscilloscope just after the acquisition with the app starts. In this case, I collect more than 100 pulses in the same 100 seconds.
Greetings

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

3 participants