

**ĐẠI HỌC QUỐC GIA HÀ NỘI**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

-----\*\*\*-----



**BÁO CÁO THỰC TẬP HỌC PHẦN**  
**THỰC TẬP NGÀNH KỸ THUẬT MÁY TÍNH**

**ĐƠN VỊ THỰC TẬP: GnodeB 5G - VHT**

**Giảng viên theo dõi: TS Trần Thị Thúy Quỳnh**

**Họ và Tên sinh viên: Luyện Huy Tín**

**Lớp: K64K2**

**Mã sinh viên: 19020636**

**Email: 19020636@vnu.edu.vn**

**Điện thoại: 0981.195.543**

**Chuyên ngành: Kỹ thuật máy tính**

**Hà Nội – 2022**

## Lời nói đầu

Đầu tiên, em xin cảm ơn đến khoa Điện Tử Viễn Thông, trường Đại học Công nghệ - Đại học Quốc Gia Hà Nội, các thầy cô đã tạo điều kiện để cho em được thực tập và làm việc trong môi trường doanh nghiệp, được thử sức trong một môi trường hoàn toàn mới, kỷ luật, nghiêm túc và cần sự cố gắng rất nhiều

Trong đợt thực tập vừa qua, bản thân em cảm thấy mình đã được học hỏi và trao dồi được rất nhiều kiến thức, cả về chuyên môn lẫn kỹ năng mềm thông qua sự chỉ dẫn và giám sát của các anh trên công ty, và cùng với đó là sự hỗ trợ đến từ phía các thầy cô ở trường khi em có thắc mắc về những vấn đề ngoài chuyên môn trong quá trình thực tập

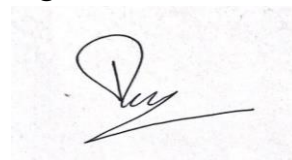
Em xin gửi lời cảm ơn đến cô Trần Thị Thúy Quỳnh, cô đã giúp đỡ và giải đáp các thắc mắc, cũng như đưa ra đường lối cho em trong kì thực tập vừa qua.

Em xin cảm ơn Trung tâm Nghiên cứu thiết bị vô tuyến băng rộng - Khối 2 – Tổng Công Ty Công nghiệp Công nghệ cao Viettel, các anh chị đã hỗ trợ và hướng dẫn em trong quá trình thực tập này, đặc biệt là anh Trưởng phòng Phần mềm hệ thống Tạ Quốc Việt, anh Tăng Thiên Vũ, anh Đậu Hồng Quân, anh Trần Hải Nam và anh Hoàng Đức Trường, trong quá trình được thực tập và làm việc đã chia sẻ cho em rất nhiều những kiến thức hay, bổ ích, để từ đó em đúc rút ra những kinh nghiệm cho bản thân, hoàn thiện bản thân hơn.

Một lần nữa, em xin chân thành cảm ơn!

*Hà Nội, ngày 29 tháng 8 năm 2022*

Người viết báo cáo



Luyện Huy Tín

# Mục Lục

<b>I. KIẾN THỨC CHUYÊN MÔN ĐẠT ĐƯỢC</b>	<b>5</b>
<b>II. KỸ NĂNG ĐẠT ĐƯỢC</b>	<b>5</b>
<b>III. MÔ TẢ CHI TIẾT QUÁ TRÌNH THỰC HIỆN CÔNG VIỆC</b>	<b>5</b>
<b>CHƯƠNG 1: LINUX PROGRAMMING ASSIGNMENT</b>	<b>6</b>
1. Nhiệm vụ được giao	6
2. Tìm hiểu về các khái niệm	6
2.1. Thread	6
2.2. Multithreading	7
2.3. Process	7
2.4. Process based và Thread based	7
2.5. Thread creation	7
2.6. Thread id	8
2.7. Thread join	8
2.8. Mutex	8
2.9. Nanosleep	9
2.10. Clock_nanosleep	9
3. Kiểm tra tính Realtime của Kernel-realtime	
3.1. Khảo sát bằng chương trình C	10
3.1.1 Thread SAMPLE	10
3.1.2. Thread INPUT	11
3.1.3. Thread LOGGING	12
3.1.4. Sử dụng Shell Script	13
3.2. Khảo sát kết quả và đưa ra đánh giá	14
3.2.1. Kết quả	14
3.2.2. Kết luận - đánh giá	19
<b>CHƯƠNG II: WORKING FLOW WITH ZYNQ CHIP</b>	<b>20</b>
1. Nhiệm vụ được giao	20
2. Tiến hành cài đặt và làm quen với các công cụ và KIT	20
2.1. Vivado	20
2.2. Petalinux	20
2.3. Kit ZCU102	20
2.4. Picozed	21
3. Xây dựng Images cho ZCU102 từ BSP	22

3.1. Cài đặt petalinux	22
3.2. Cài đặt môi trường	22
3.3. Tạo project từ file BSP	23
3.4. Xây dựng hệ thống Images cho mạch	24
3.5. Tạo file BOOT.BIN	24
3.6. Tiến hành copy file sang SD card	25
3.7. Cài đặt Picocom	27
3.8. Boot cho mạch ZCU102	27
4. Chạy Thread trên KIT ZCU102	30
4.1. Chuẩn bị phần cứng	30
4.2. Kiểm tra IP address	31
4.3. SSH máy tính đến Server	32
4.4. Cài đặt và thực hiện cross compile	32
4.5. Chuyển file đã compile sang KIT	33
4.6. Kết quả thu được	33
4.6.1. Khi chạy độc lập	34
4.6.2. Kết luận - đánh giá	33
4.6.3. Khi chạy có tải( file disturb)	35
4.6.4. Kết luận - đánh giá	37
5. Xây dựng images từ file Hardware tự Design cho KIT ZCU102	41
5.1. Sử dụng vivado để design IP	41
5.2. Xuất ra file Hardware	47
5.3. Config cho project	50
5.4. Build image và boot cho ZCU102	52
6. Tổng kết quá trình Booting cho KIT	55
6.1. Bootloader	55
6.2. Kernel	56
6.3. User	57

## **I. KIẾN THỨC CHUYÊN MÔN ĐẠT ĐƯỢC**

- Hiểu được về đa luồng, cách sử dụng và tác dụng của lập trình thời gian thực.
- Được làm việc trên server, sử dụng shell script, lập trình C trên Linux
- Hiểu sâu và rõ toàn bộ workflow khi xây dựng OS cho KIT Xilinx
- Học được cách sử dụng các công cụ để tạo ra được thiết kế mong muốn ( tạo các IP core, cách nối các IP, ...)
- Xây dựng được image cho KIT từ file hardware tự tạo, và file BSP của nhà sản xuất
- Biết được thêm nhiều các khái niệm liên quan đến việc booting cho KIT( FSBL, U-boot, ...)
- Được tìm hiểu về device driver

## **II. KỸ NĂNG ĐẠT ĐƯỢC**

- Kỹ năng teamwork khi thực hiện công việc
- Học được cách tự đặt ra vấn đề và tự mình giải quyết nó
- Cách sử dụng keyword để tìm những tài liệu cần thiết
- Cải thiện thêm về trình độ ngoại ngữ
- Khả năng giao tiếp được cải thiện
- Tự đánh giá được năng lực của bản thân

## **III. MÔ TẢ CHI TIẾT QUÁ TRÌNH THỰC HIỆN CÔNG VIỆC**

## CHƯƠNG 1: LINUX PROGRAMMING ASSIGNMENT

### 1. Nhiệm vụ được giao

- Viết chương trình C trên Linux chạy 3 thread SAMPLE, LOGGING, INPUT. Trong đó:
  - + Thread SAMPLE thực hiện vô hạn lần nhiệm vụ sau với chu kỳ X ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến T.
  - + Thread SAMPLE thực hiện vô hạn lần nhiệm vụ sau với chu kỳ X ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến T.
  - + Thread SAMPLE thực hiện vô hạn lần nhiệm vụ sau với chu kỳ X ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến T.
- Viết shell script để thay đổi lại giá trị chu kỳ X trong file “freq.txt” sau mỗi 1 phút. Các giá trị X lần lượt được ghi như sau: 1000000 ns, 100000 ns, 10000 ns, 1000 ns, 100ns.
- Chạy shell script + chương trình C trong vòng 5 phút, sau đó dừng chương trình C.
- Thực hiện khảo sát file “time\_and\_interval.txt”: Vẽ đồ thị giá trị interval đối với mỗi giá trị chu kỳ X và đánh giá.

### 2. Tìm hiểu về các khái niệm

#### 2.1. Thread

- Là một đơn vị cơ bản trong CPU. Một luồng sẽ chia sẻ với các luồng khác trong cùng process về thông tin data, các dữ liệu của mình. Việc tạo ra thread giúp cho các chương trình có thể chạy được nhiều công việc cùng một lúc.

## 2.2. Multithreading

- Đa luồng là khả năng của một chương trình hoặc một hệ điều hành cho phép nhiều người dùng cùng một lúc mà không yêu cầu nhiều bản sao của chương trình đang chạy trên máy tính. Đa luồng cũng có thể xử lý nhiều yêu cầu từ cùng một người dùng. Chúng ta có thể chia ra làm 2 loại multitasking: Process-based và Thread-based tương ứng: dựa trên tiến trình(process) và dựa trên luồng(thread).

## 2.3. Process

- Process là một chương trình đang được thực thi (đang chạy). Nhưng, một chương trình không phải là một process. Vì chương trình là một file, hay một folder bị động nằm trên máy. Trong khi đó, một process là một chương trình đang hoạt động (đang chạy, đã được tải lên bộ nhớ chính để hoạt động).
- Một chương trình có thể có hai (hay nhiều) process đang chạy, nhưng chúng được coi là hai (hay nhiều) quá trình độc lập với nhau.

## 2.4. Process based và Thread based

- Process based cho phép hệ thống thực thi hai hoặc nhiều chương trình đồng thời.
- Thread based cho phép một chương trình thực hiện hai hoặc nhiều tác vụ dưới dạng luồng đồng thời.

## 2.5. Thread creation

- Khi chương trình chính bắt đầu, nó được gọi là một Thread. Thread điều khiển hàm main() được gọi là main thread. Các Thread khác do tiến trình tạo ra sau đó được gọi là Thread phụ. Mỗi Thread được cung cấp cho một số định danh gọi là thread ID. Để tạo ra một Thread mới ngoài main Thread, bạn gọi hàm pthread\_create(). Hàm này được khai báo như sau:

```
#include <pthread.h>

Int pthread_create ( pthread_t * thread,
                    pthread_attr_t * attr,
                    void* (*start_routine) (void*),
                    void* arg
```

- Hàm `pthread_create()` nhận 4 tham số, tham số thứ nhất có kiểu cấu trúc `pthread_t` để lưu các thông tin về tuyến sau khi tạo ra. Tham số thứ hai dùng để đặt thuộc tính cho thread (trong trường hợp ta đặt giá trị `NULL` thì thread được tạo ra với các thuộc tính mặc định). Tham số thứ ba là địa chỉ của hàm mà thread sẽ dùng để thực thi. Tham số thứ tư là địa chỉ đến vùng dữ liệu sẽ truyền cho hàm thực thi thread.

## 2.6. Thread id

- Mỗi một thread trong process được xác định bởi một Thread ID. thread ID không nhất thiết phải là một giá trị số nguyên. để in ra Thread ID ta có thể dùng câu lệnh

```
#include <pthread.h>
pthread_t pthread_self(void);
```

## 2.7. Thread join

- Được sử dụng để đợi cho việc kết thúc của một thread hoặc chờ để tái kết hợp với luồng chính của hàm main. Sau khi create, main của thread sẽ chạy ngay lập tức và Join sẽ đợi thread return xong mới tiếp tục chạy.

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **retval);
```

## 2.8. Mutex

- Một trong những vấn đề quan tâm hàng đầu của việc điều khiển lập trình đa luồng trong cùng không gian địa chỉ của tiến trình đó là đồng bộ hóa. Bạn phải đảm bảo được nguyên tắc các thread không bị xung đột.. Để giải quyết tranh chấp và xử lý đồng bộ hóa chúng ta sử dụng một khái niệm gọi là mutex.
- Để tạo ra đối tượng mutex, trước hết cần khai báo biến kiểu cấu trúc `pthread_mutex_t`, đồng thời khởi tạo giá trị ban đầu cho biến này. Cách đơn giản nhất để khởi tạo cấu trúc mutex là dùng hằng định nghĩa trước `PTHREAD_MUTEX_INITIALIZER`. Mã khai báo mutex thường có dạng sau:

```
pthread_t a_mutex = PTHREAD_MUTEX_INITIALIZER
```



- Sau khi sử dụng xong mutex bạn nên hủy nó. Sử dụng xong có nghĩa là không còn tuyến nào cần chiếm giữ mutex cho các cho tác khóa/tháo khóa nữa. Hàm `pthread_mutex_destroy()` được dùng để hủy mutex.

```
rc = pthread_mutex_destroy (&a_mutex)
```

## 2.9. Nanosleep

- Hàm `nanosleep()` thực hiện một nhiệm vụ tương tự như `sleep()`, nhưng cung cấp một số lợi thế, bao gồm độ chính xác tốt hơn khi chỉ định khoảng thời gian ngủ.

```
#include <time.h>
int nanosleep(const struct timespec *request, struct timespec *remain);

struct timespec {
    time_t tv_sec;    /* seconds */
    long   tv_nsec    /* nanoseconds */
}
```

biến `tv_nsec` chỉ định một giá trị nano giây. Nó phải là một số trong phạm vi 0 đến 999.999.999.

## 2.10. Clock\_nanosleep

- `Clock_nanosleep` cho phép thread gọi ngủ cho một khoảng thời gian được chỉ định với độ chính xác nano giây. Nó khác nhau trong việc cho phép người gọi chọn đồng hồ mà khoảng thời gian ngủ sẽ được đo và cho phép khoảng thời gian ngủ được chỉ định là giá trị tuyệt đối hoặc tương đối.

```
#include <time.h>
int clock_nanosleep(clockid_t clockid, int flags, const struct timespec
*request, struct timespec *remain);

struct timespec {
    time_t tv_sec;    /* seconds */
    long   tv_nsec    /* nanoseconds */
}
```

### 3. Kiểm tra tính real-time của linux kernel-realtime

#### 3.1. Khảo sát bằng chương trình C

##### 3.1.1 Thread SAMPLE

- Yêu cầu bài toán:
  - + Thread SAMPLE thực hiện vô hạn lần nhiệm vụ sau với chu kì X ns. Nhiệm vụ là đọc thời gian hệ thống hiện tại (chính xác đến đơn vị ns) vào biến T.
- Hướng giải quyết:
  - + Sử dụng hàm nanosleep() hoặc clock\_nanosleep() để đọc giá trị thời gian
  - + Tạo 1 định danh, sau đó viết hàm main cho thread sample có tên là void \*currently\_time(Hàm này trả về giá trị thời gian thực chính xác đến từng s và ns). Sau đó, ta sử dụng hàm nanosleep hoặc clock\_nanosleep để đọc giá trị thời gian theo biến chu kì T ( từ thread INPUT)
- Trình bày kết quả:

```
void *currently_time(void *time)
{
    clock_gettime(CLOCK_REALTIME, &request);
    while(1)
    {

        request.tv_nsec += cycle;
        /* xu ly tran */
        if(request.tv_nsec > 1000*1000*1000) {
            request.tv_nsec -= 1000*1000*1000;
            request.tv_sec
            ++;
        }
        clock_nanosleep(CLOCK_REALTIME,
            TIMER_ABSTIME, &request, NULL);

        clock_gettime(CLOCK_REALTIME, &now);

    }
    return NULL;
}
```

```
}
```

### 3.1.2 Thread INPUT

- Yêu cầu bài toán:
  - + Kiểm tra file “freq.txt” để xác định chu kỳ X (của thread SAMPLE) có bị thay đổi không?, nếu có thay đổi thì cập nhật lại chu kỳ X. Người dùng có thể echo giá trị chu kỳ X mong muốn vào file “freq.txt” để thread INPUT cập nhật lại X.
- Hướng giải quyết:
  - + Tạo 1 file freq.txt để lưu giá trị chu kỳ mình muốn từ ban đầu
  - + Tạo 1 hàm check\_time để đọc file freq.txt.
  - + Sử dụng hàm strtol để convert giá trị string sang long, từ đó giá trị sẽ được thread SAMPLE đọc và xử lý.
- Trình bày kết quả:

```
void *check_time(void *time)
{
    while(1)
    {
        // struct timespec timecheck;
        FILE *file;
        file = fopen("freq.txt","r");
        char buff[100];
        fgets(buff,sizeof(buff),file);
        //convert from string to long
        char *eptr;
        cycle = strtol(buff,&eptr,10);
        fclose(file);
        return NULL;
    }
}
```

- Hàm strtol chuyển đổi phần ban đầu của chuỗi trong str thành giá trị int dài theo cơ sở đã cho, giá trị này phải nằm trong khoảng từ 2 đến 36, hoặc là giá trị đặc biệt 0.

```
Long int strtol(const char *str, char **endptr, int base)
```

### 3.3. Thread LOGGING

- Yêu cầu bài toán:
  - + Chờ khi biến T được cập nhật mới, thì ghi giá trị biến T và giá trị interval (offset giữa biến T hiện tại và biến T của lần ghi trước) ra file có tên “time\_and\_interval.txt”
- Hướng giải quyết:
  - + Thực hiện mở file save\_value.txt ở chế độ a+ (đọc và ghi vào cuối file, nếu file chưa tồn tại sẽ thực hiện tạo 1 file mới)
  - + Tạo 2 biến old (1 biến có độ chính xác là s và 1 biến là ns) sử dụng hàm strtol để convert chuỗi trong file save\_value sang kiểu long.
  - + Thực hiện so sánh với giá trị now(so sánh cả giá trị s và ns), sau đó tính offset thông qua chênh lệch giữa các biến old và now
  - + Lưu giá trị mới vào file save\_value
- Trình bày kết quả:

```
void *save_time(void *time)
{
    FILE *file1;
    while(1){

        file1 = fopen("save_value.txt","a+"); // save all of offset
        values in file

        if(now.tv_nsec != old.tv_nsec || now.tv_sec != old.tv_sec) {
            long interval_sec = ((long)now.tv_sec) - (long)old.tv_sec ;
            long interval_nsec;

            if(now.tv_nsec > old.tv_nsec)
            {
                interval_nsec = now.tv_nsec - old.tv_nsec;
            }
            else
            {
                interval_nsec = 1000000000 + now.tv_nsec - old.tv_nsec;
```

```

        interval_sec = interval_sec - 1;
    }

    old.tv_nsec = now.tv_nsec;
    old.tv_sec = now.tv_sec;

    fprintf(file1,"%ld \n",interval_nsec);
    fclose(file1);
}

// sleep(1);
}

return NULL;
}

```

### 3.1.4. Sử dụng Shell Script

- Yêu cầu bài toán:
  - + Viết shell script để thay đổi lại giá trị chu kỳ X trong file “freq.txt” sau mỗi 1 phút. Các giá trị X lần lượt được ghi như sau: 1000000 ns, 100000 ns, 10000 ns, 1000 ns, 100ns.
- Hướng giải quyết:
  - + Dùng lệnh echo để thực hiện ghi đè vào file freq.txt các giá trị lần lượt là 1000000ns, 100000ns, 10000ns, 1000ns, 100ns. Với mỗi giá trị ta thực hiện chạy timeout trong vòng 60s. Tổng cộng hết 5 phút
- Trình bày kết quả:

```

echo “1000000” > freq.txt
timeout 60s ./test
echo “100000” > freq.txt
timeout 60s ./test
echo “10000” > freq.txt
timeout 60s ./test
echo “1000” > freq.txt
timeout 60s ./test
echo “100” > freq.txt
timeout 60s ./test

```

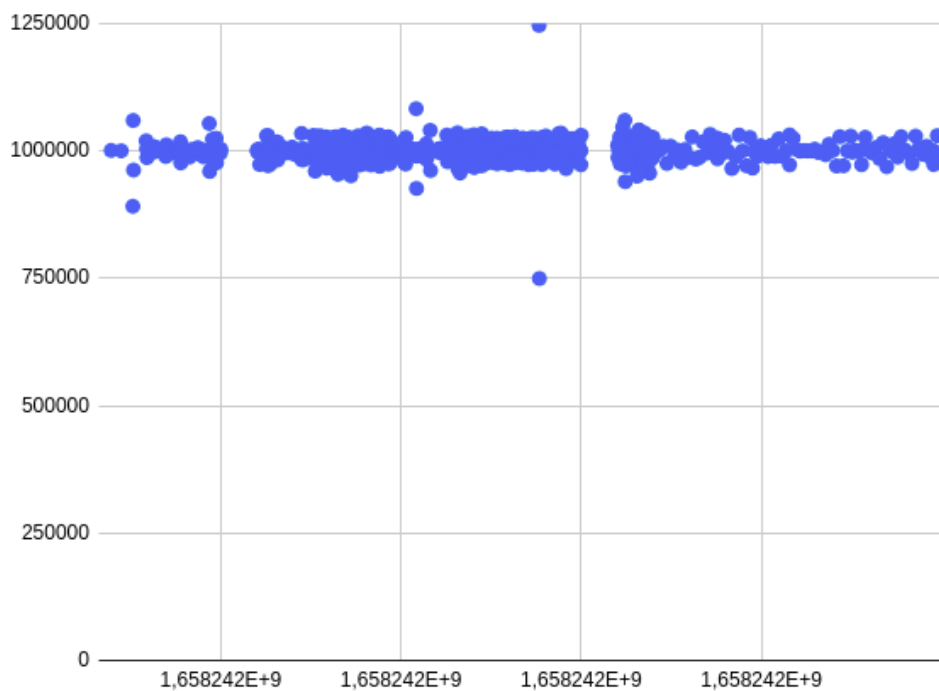
## 4. Khảo sát kết quả và đưa ra đánh giá

- Thực hiện vẽ đồ thị histogram với mỗi giá trị interval lưu trong file. Đánh giá và đưa ra kết luận.

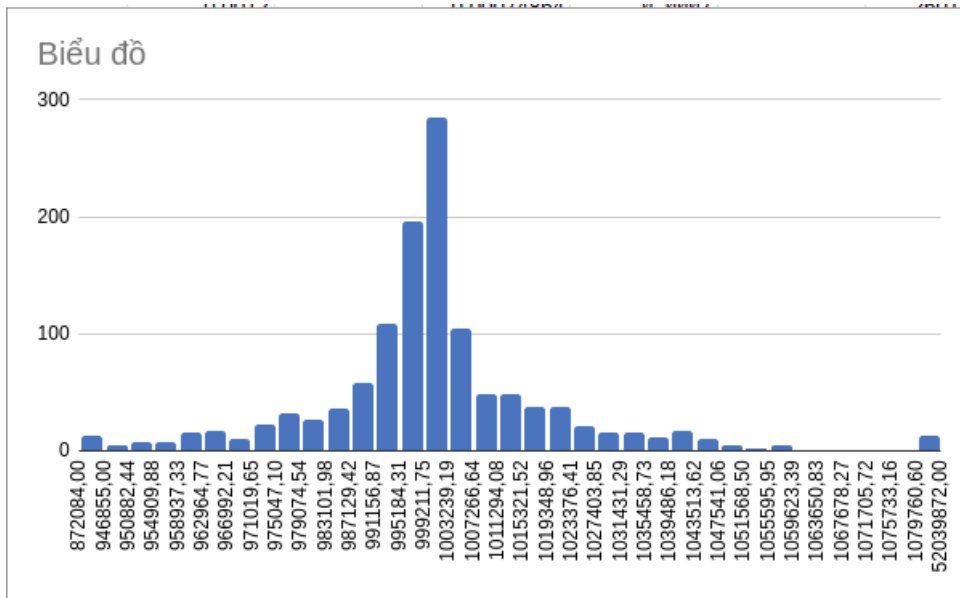
### 4.1. Kết quả

\* *Chú thích:*

- *Đối với đồ thị biến: Ta có trục hoành là lượng giá trị biến, trục tung là giá trị chu kì*
- *Đối với đồ thị phân bố của biến: Ta có trục hoành là giá trị lấy mẫu thu được, trục hoành là số lượng giá trị đã lấy mẫu*
- **Khảo sát khi chu kì bằng 1000000ns**



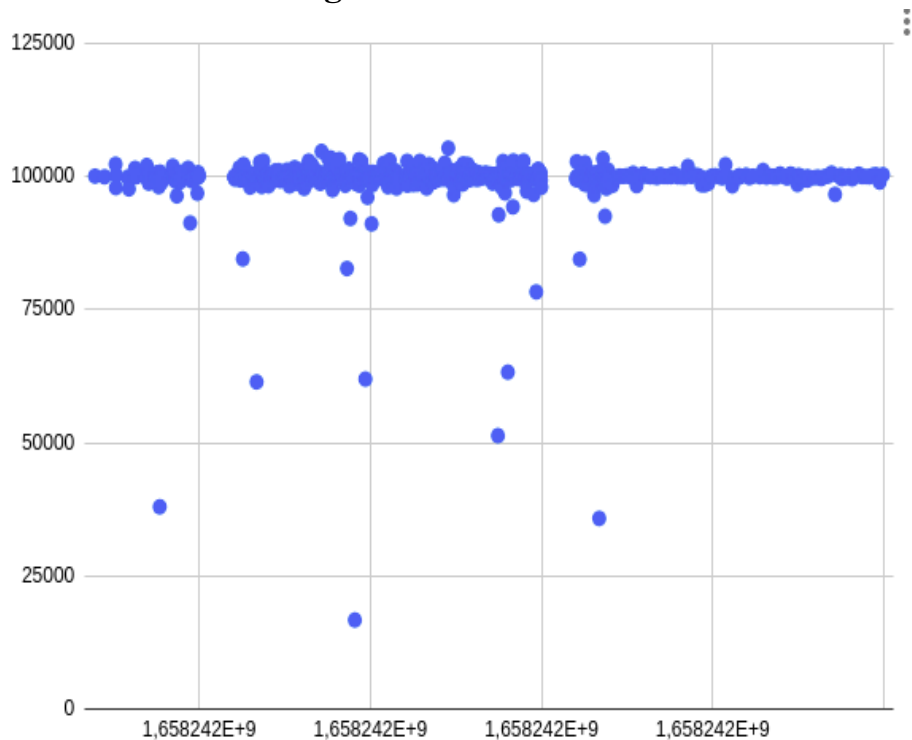
Hình 1.2. Đồ thị biến khi  $T = 1000000\text{ns}$



Hình 1.3. Đồ thị phân bố của biến khi  $T = 1000000ns$

- Nhận xét: ta có giá trị offset giao động quanh mức 995184ns – 1003239ns, các biến giao động ổn định. Giá trị ta thu được gần như trùng khớp với giá trị  $T$  mà ta đã đưa ra

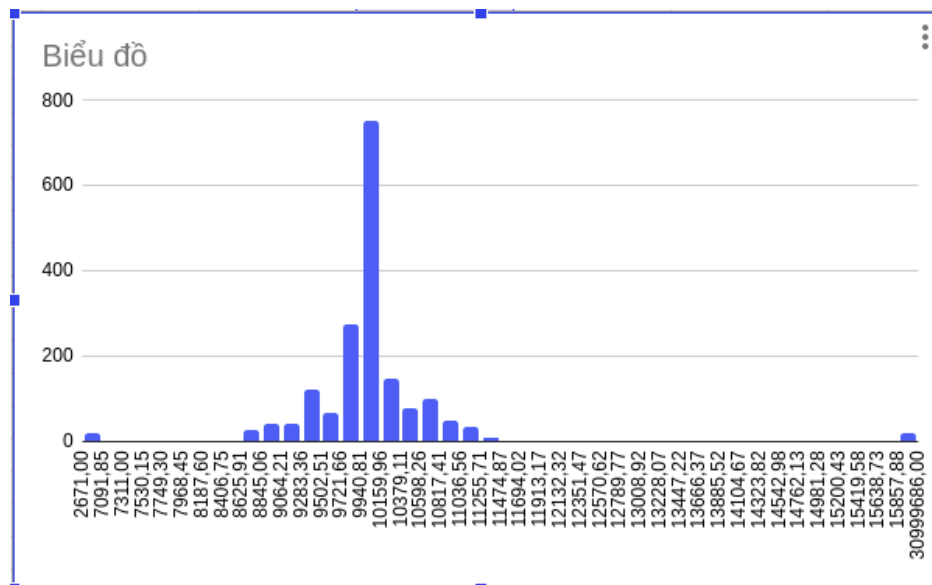
- **Khảo sát khi chu kì bằng 100000ns**



Hình 1.4. Đồ thị biến khi  $T = 100000ns$



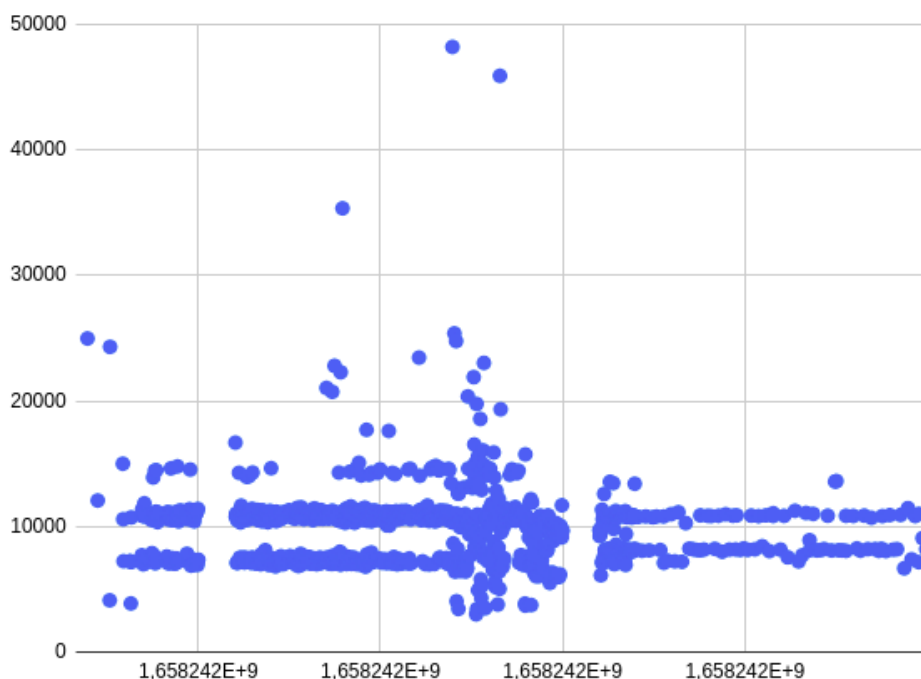




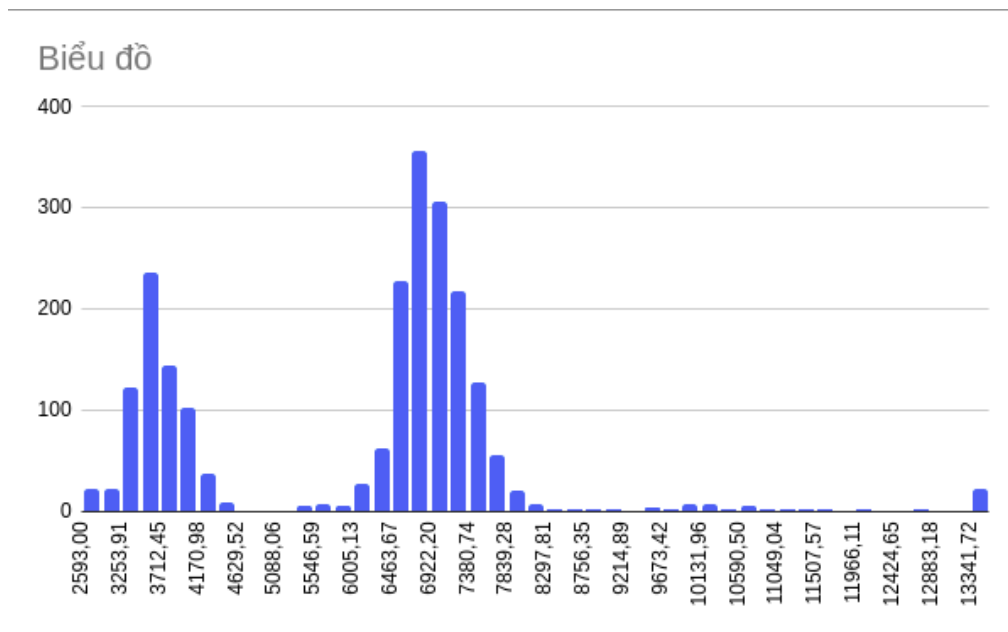
Hình 1.7. Đồ thị phân bố của biến khi  $T = 10000\text{ns}$

Nhận xét: ta có giá trị offset giao động quanh mức  $9940\text{ns} - 10159\text{ns}$ , giá trị vẫn được coi là ổn định vì chủ yếu giao động quanh mức  $T$  đã đặt trước là  $10000\text{ns}$ . Tuy nhiên dễ thấy, sai số càng ngày càng lớn hơn

#### - Khảo sát khi chu kì bằng $1000\text{ns}$



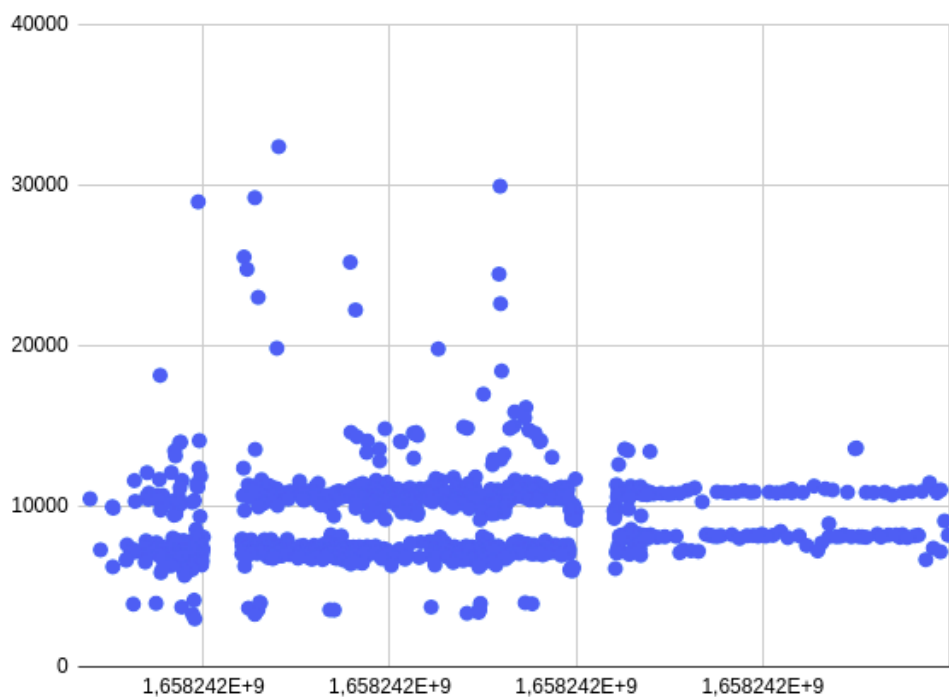
Hình 1.8. Đồ thị biến khi  $T = 1000\text{ns}$



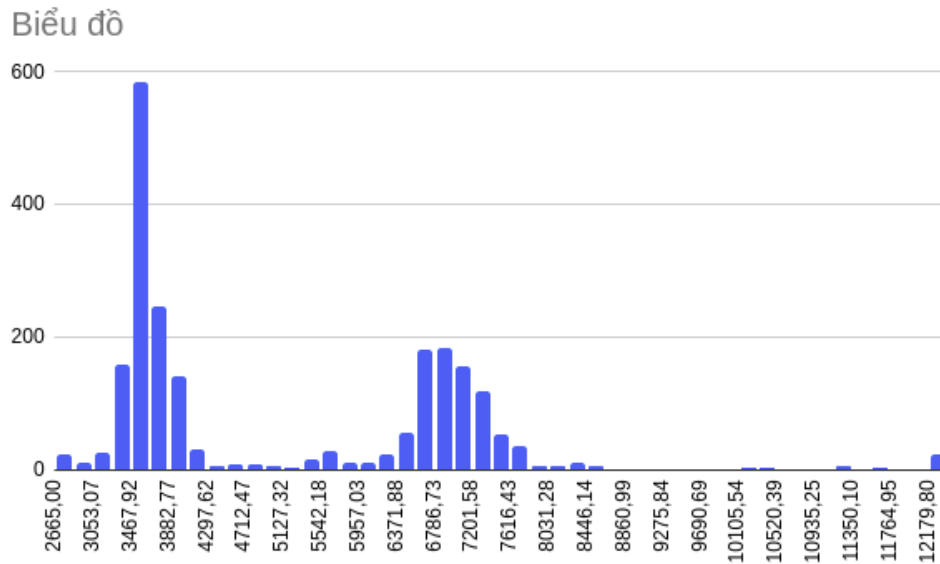
Hình 1.9. Đồ thị phân bố của biến khi  $T = 1000$

Nhận xét: ta thấy giá trị offset giao động quanh mức 3253ns-3712ns và 6463ns-7380 ns, lúc này ta thấy rằng giá trị offset đã không còn chính xác như những trường hợp trên

#### - Khảo sát khi chu kì bằng 100ns



Hình 1.10. Đồ thị biến khi  $T = 100\text{ns}$



Hình 1.11. Đồ thị phân bố của biến khi  $T = 100ns$

Nhận xét: ta thấy giá trị offset giao động quanh mức 3467ns – 3882ns và 6786ns - 7201ns

## 4.2. Kết luận - đánh giá

- Ta có thể thấy mục đích của bài toán muốn ta khảo sát hiệu năng của CPU đối với việc lấy mẫu các giá trị từ lớn xuống bé. Từ đó đánh giá được sức mạnh của máy tính
- Với giá trị chu kỳ bằng 1000000ns, 100000ns, 10000ns, thông qua đồ thị biến và đồ thị phân bố của biến, dễ dàng nhận thấy độ ổn định của việc lấy mẫu là rất tốt, khi giá trị lấy mẫu trả về luôn phân bố xung quanh giá trị chu kỳ (giá trị được ta setup cho biến chu kỳ ngay từ ban đầu). Qua đó cho thấy rằng CPU hoạt động ổn định và không xảy ra biến số nhiều khi  $T$  ở khoảng cao
- Tuy nhiên đối với giá trị chu kỳ bằng 1000ns, 100ns, cũng từ những biểu đồ ta thấy rằng, những giá trị lấy mẫu thu được đã không thể chính xác 100% như những lần kiểm tra trước đó. Điều này cho thấy rằng, hiệu năng của CPU đã tới giới hạn. Ở đây, cụ thể giới hạn lấy mẫu giá trị của CPU giao động trong khoảng [2500ns-3000ns]
- Như vậy có thể khẳng định rằng, Chu kỳ càng cao khi giá trị trả về càng chính xác. Không chỉ cần phải tối ưu cho chương trình, tìm đọc hiểu thêm tài liệu, mà hiệu năng của CPU cũng rất quan trọng trong những bài toán mà ta muốn làm việc với hệ thống real-time

## CHƯƠNG II: WORKING FLOW WITH ZYNQ CHIP

### 1. Nhiệm vụ được giao

- Mô tả quá trình được làm việc với các công cụ, KIT của Xilinx. Cụ thể như vivado, petalinux, ZUC102, Picozed, để có thể porting được Linux cho KIT đó

### 2. Tiến hành cài đặt và làm quen với các công cụ và KIT

#### 2.1. Vivado

- Cài đặt vivado 2017.4 cho Ubuntu
- Thực hiện các bài test để làm quen với việc sử dụng Vivado. Cụ thể Adding IP cores in PL, Writing Basic Software Application, ...
- Link cài đặt: <https://www.xilinx.com/support/download.html>
- Link tài liệu: <https://www.xilinx.com/support/university/workshops.html>

#### 2.2. Petalinux

- Cài đặt petalinux 2020.1 cho Ubuntu
- Tiến hành cài đặt môi trường, tạo project theo template, build thử images dựa theo file BSP do nhà sản xuất cung cấp
- link cài đặt:  
<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html>
- link tài liệu: <https://docs.xilinx.com/v/u/2020.1-English/ug1144-petalinux-tools-reference-guide>

#### 2.3. Kit ZCU102

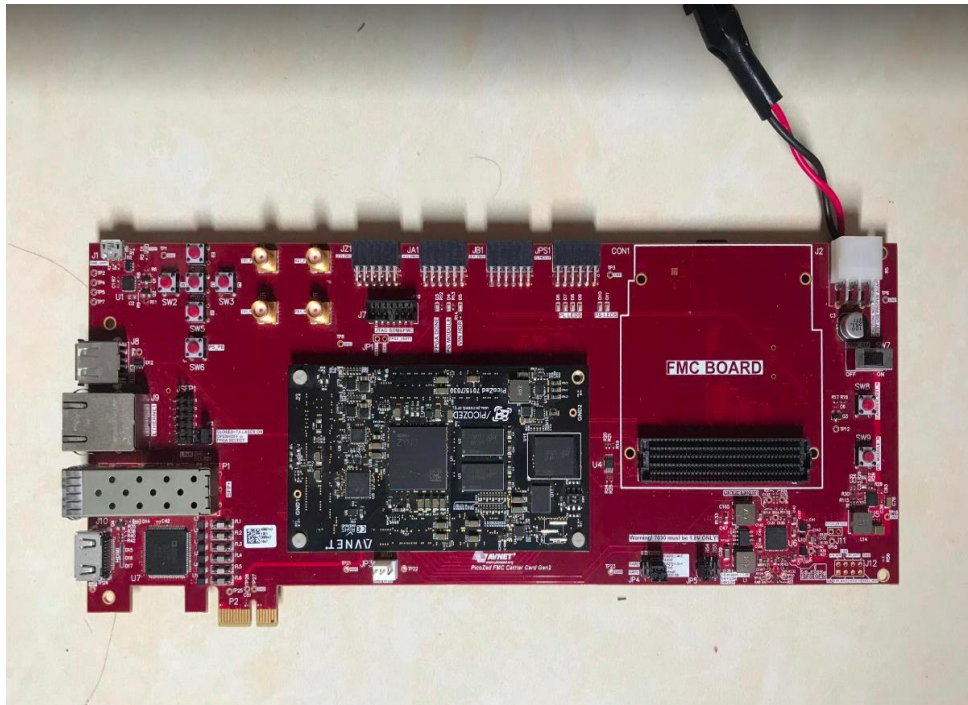
- ZCU102 Evaluation Kit cho phép các nhà thiết kế bắt đầu thiết kế cho các ứng dụng ô tô, công nghiệp, video và truyền thông. Kit sử dụng Zynq® UltraScale+™ MPSoC với một Arm® Cortex®-A53 quad-core, dual-core Cortex-R5F bộ xử lý thời gian thực, và một Mali™-400 MP2 đơn vị xử lý đồ họa dựa trên Xilinx's 16nm FinFET+ programmable logic fabric. ZCU102 hỗ trợ tất cả các thiết bị ngoại vi và giao diện chính, cho phép phát triển một loạt các ứng dụng.
- Link Tài liệu:  
[https://www.xilinx.com/support/documents/boards\\_and\\_kits/zcu102/ug1182-zcu102-eval-bd.pdf](https://www.xilinx.com/support/documents/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf)



Hình 2.1. Bo mạch ZCU102

#### 2.4. Picozed

- Picozed chứa các yêu cầu cốt lõi để hỗ trợ thiết kế SOC bao gồm bộ nhớ, cấu hình, ethernet, USB và đồng hồ. Nó cung cấp quyền truy cập dễ dàng vào hơn 100 chân I/O của người dùng thông qua ba đầu nối I/O ở mặt sau của mô-đun. Các đầu nối này có thể hỗ trợ các giao diện chuyên dụng cho Ethernet, USB, JTAG, nguồn điện và các tín hiệu điều khiển khác, cũng như các bộ thu phát GTP/GTX trên các mẫu 7015/7030. Bộ thu phát dựa trên 7015 và 7030 phiên bản picoZed là một siêu của phiên bản 7010/7020, thêm bốn cổng thu phát nối tiếp tốc độ cao vào đầu nối I/O. Bạn có thể thiết kế thẻ vận chuyển của riêng mình, cắm một picozed và bắt đầu phát triển ứng dụng với hệ thống con ZYNQ-7000 AP SOC đã được chứng minh.
- Link tài liệu: <https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/picozed/>



Hình 2.2. Bo mạch Picozed fmc carrier card V2

### 3. Xây dựng Images cho ZCU102 từ BSP

#### 3.1. Cài đặt petalinux

- Sau khi tải petalinux-version.run (link tải ở trên) thành công, ta thực hiện câu lệnh như dưới để có thể install được petalinux

```
./petalinux-v<petalinux-version>-final-installer.run
```

#### 3.2. Cài đặt môi trường

- Sau khi cài đặt thành công petalinux, ta cần cd đến thư mục chứa petalinux, thực hiện lệnh ls để kiểm tra file setting.sh, sau đó thực hiện câu lệnh dưới đây

For Bash as user login shell:

```
$ source <path-to-installed-PetaLinux>/settings.sh
```

For C shell as user login shell:

```
$ source <path-to-installed-PetaLinux>/settings.csh
```

### 3.3. Tạo project từ file BSP


- Tải file BSP của ZCU102 từ nhà sản xuất. BSP là Board Support Package. Để kernel chạy được trên phần cứng, thì phải có BSP (Board Support Package).
- BSP chứa các phần giao tiếp với phần cứng cụ thể, nó đóng vai trò trung gian cho hầu hết các lớp khác của Kernel và phần cứng.
- BSP có thể từ mainstream (tức là được tích hợp trong source của nhân Linux) hoặc từ chính các Vendor tạo ra phần cứng (các board).
- link:


<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html>


#### Zynq UltraScale+ MPSoC Board Support Packages - 2020.1

**Important Information**

Download only the required BSP(s) depending on the evaluation board that is being used. All BSPs have a prebuilt directory with bootable images. Hover your mouse over the download hyper-link to see a description of the BSP contents.

 **ZCU102 BSP** (BSP - 450.57 MB)  
MD5 SUM Value : e9e9ae680392d2ec53fcbb5b0aa470b9

 **ZCU104 BSP** (BSP - 1.06 GB)  
MD5 SUM Value : 53e6b2166f33d80fa941f3078a2dd85d

 **ZCU106 BSP** (BSP - 1.04 GB)  
MD5 SUM Value : bde26bc3d52d760c8386147953c7b392

Hình 3.1 File BSP do nhà sản xuất cung cấp

- Sau khi tải thành công file .BSP ta tiến hành di chuyển file vào thư mục lưu petalinux, và thực hiện câu lệnh dưới đây để tạo project



Run `petalinux-create` command on the command console:

```
petalinux-create -t project -s <path-to-bsp>
```

### 3.4. Xây dựng hệ thống Images cho mạch

- Sau khi hoàn thành các bước trên, ta cd vào thư mục project đã tạo và thực hiện câu lệnh dưới đây để build images cho mạch

Run `petalinux-build` to build the system image:

```
$ petalinux-build
```

- Quá trình sẽ hiển thị như bên dưới

```
petalinux-build
INFO: sourcing build tools
[INFO] building project
[INFO] generating Kconfig for project
[INFO] silentconfig project
[INFO] extracting yocto SDK to components/yocto
[INFO] sourcing build environment
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating plnxtool conf
[INFO] generating user layers
[INFO] generating workspace directory
INFO: bitbake petalinux-image-minimal
Parsing recipes: 100% |
#####
#####| Time:
0:00:35
Parsing of 2961 .bb files complete (0 cached, 2961 parsed). 4230 targets,
168 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Fetching uninative binary shim from file:///scratch/xilinx-
zcu102-2020.1/components/yocto/downloads/uninative/
9498d8bba047499999a7310ac2576d0796461184965351a56f6d32c888a1f216/x86_64-
nativesdk-
libc.tar.xz;sha256sum=9498d8bba047499999a7310ac2576d0796461184965351a56f6d32
c888a1f216
Initialising tasks: 100% |
#####
#####| Time: 0:00:03
Checking sstate mirror object availability: 100% |
#####
#####| Time: 0:00:21
Sstate summary: Wanted 1016 Found 803 Missed 213 Current 0 (79% match, 0%
complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: Tasks Summary: Attempted 3614 tasks of which 2619 didn't need to be
rerun and all succeeded.
INFO: Failed to copy built images to tftp dir: /tftpboot
[INFO] successfully built project
```

Hình 3.2. Quá trình xây dựng images

- Sau khi terminal hiển thị như trên, nghĩa là chúng ta đã build Images thành công cho mạch.

### 3.5. Tạo file BOOT.BIN

- Sau khi đã build thành công, sẽ xuất hiện thư mục Images như dưới



```
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~/petalinux/xilinx-zcu102-2020.1$ ls -l
total 44
drwxrwxr-x 8 tindz tindz 4096 Aug  8 23:15 build
drwxr-xr-x 4 tindz tindz 4096 Aug  8 15:30 components
-rw-r--r-- 1 tindz tindz 248 May 28 2020 config.project
drwxr-xr-x 3 tindz tindz 4096 May 28 2020 hardware
drwxrwxr-x 3 tindz tindz 4096 Aug  8 18:35 images
drwxr-xr-x 3 tindz tindz 4096 May 28 2020 pre-built
drwxr-xr-x 5 tindz tindz 4096 May 28 2020 project-spec
-rw-r--r-- 1 tindz tindz 14929 May 28 2020 README
```

Hình 3.3. Hiển thị các thư mục quan trọng có project

- Tiến hành cd vào /images/linux, thực hiện lệnh ls để xem các file có trong thư mục

```
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~/petalinux/xilinx-zcu102-2020.1/images$ ls
linux
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~/petalinux/xilinx-zcu102-2020.1/images$ cd linux/
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~/petalinux/xilinx-zcu102-2020.1/images/linux$ ls
bl31.bin  pnu-fw.elf  rootfs.manifest  vmlinux
bl31.elf  pxelinux.cfg  rootfs.tar.gz    zynqmp_fsbl.elf
BOOT.BIN  rootfs.cpio  system.bit       zynqmp-qemu-arm.dtb
boot.scr  rootfs.cpio.gz  system.dtb       zynqmp-qemu-multiarch-arm.dtb
Image     rootfs.cpio.gz.u-boot  u-boot.bin       zynqmp-qemu-multiarch-pmu.dtb
image.ub  rootfs.jffs2  u-boot.elf
```

Hình 3.4. Hiển thị các file có trong thư mục images

- Sau khi thấy file zynqmp\_fsbl và system.bit, thực hiện lệnh như dưới:  
Follow the step below to generate the boot image in .BIN format.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

- Lúc này thư mục sẽ xuất hiện file BOOT.BIN, điều đó cho thấy ta đã generate thành công.

### 3.6. Tiến hành copy file sang SD card

- Để có thể boot được cho mạch, ta cần copy file BOOT.BIN và image.ub sang SD card, tuy nhiên để copy được, đầu tiên ta cần set phân vùng cho sd card.
- Sau khi cắm thẻ SD vào máy tính. thực hiện lệnh như dưới để ngắt kết nối với thẻ SD

```
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~$ umount /media/tindz/BOOT
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~$ umount /media/tindz/rootfs
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~$
```

- Tiếp đó ta thực hiện lệnh fdisk để cài đặt phân vùng cho sd card

```
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~$ sudo fdisk /dev/mmcblk2

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): m
```

```

DOS (MBR)
a  toggle a bootable flag
b  edit nested BSD disklabel
c  toggle the dos compatibility flag

Generic
d  delete a partition
F  list free unpartitioned space
l  list known partition types
n  add a new partition
p  print the partition table
t  change a partition type
v  verify the partition table
i  print information about a partition

Misc
m  print this menu
u  change display/entry units
x  extra functionality (experts only)

Script
I  load disk layout from sfdisk script file
O  dump disk layout to sfdisk script file

Save & Exit
w  write table to disk and exit
q  quit without saving changes

Create a new label
g  create a new empty GPT partition table
G  create a new empty SGI (IRIX) partition table
o  create a new empty DOS partition table
s  create a new empty Sun partition table

```

Hình 3.5. Các lựa chọn để cài đặt cho phân vùng SD card

- Kiểm tra phân vùng sau khi đã edit thành công

```

Command (m for help): p
Disk /dev/mmcblk2: 7.41 GiB, 7948206080 bytes, 15523840 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x34eb5e95

Device            Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk2p1    *             2048   2099199   2097152    1G 83 Linux
/dev/mmcblk2p2                2099200 15523839 13424640   6.4G 83 Linux

```

Hình 3.6. Hiện thị phân vùng hiện có

- Sau khi đã cài đặt phân vùng thành công ta thực hiện format cho phân vùng như dưới đây

#### Format the partitions

- Format 1st partition by using fat.
    - `mkfs.vfat -F 32 -n BOOT /dev/<first partition>`
  - Format the 2nd partition by using ext2/ext4.
    - `mkfs.ext4 -L rootfs /dev/<second partition>`
- Sau khi format thành công, cuối cùng ta cần copy file từ thư mục sang sd card. Sử dụng câu lệnh cp như dưới đây. Copy sang cả 2 phân vùng BOOT và rootfs

```
aslam@aslam-pc:~/Desktop/IDLAB/petalinux/projects/zed/zedboard$ cp images/linux/BOOT.BIN /media/aslam/BOOT/
aslam@aslam-pc:~/Desktop/IDLAB/petalinux/projects/zed/zedboard$ cp images/linux/image.ub /media/aslam/BOOT/
```

### 3.7. Cài đặt Picocom

- Ta cần cài đặt picocom để truy console Linux dựa trên cổng nối tiếp; thường được thực hiện khi phát triển một sản phẩm dựa trên Linux nhúng. Cụ thể ở đây là kiểm tra xem hệ điều hành có được boot thành công không. Câu lệnh để cài đặt:

```
sudo apt-get install picocom
```

### 3.8. Boot cho mạch ZCU102

- Đầu tiên ta cần cắm dây USB với cổng UART của mạch ZCU102, điều chỉnh các jump và sw về đúng vị trí mà ta mong muốn thực hiện

Jumper	Function	Default
J85	POR_OVERRIDE • 1-2: Enable • 2-3: Disable	2-3
J12	SYSMON I2C Address • Open: SYSMON_VP_R floating • 1-2: SYSMON_VP_P pulled down	1-2
J13	SYSMON I2C Address • Open: SYSMON_VN_R floating • 1-2: SYSMON_VP_N pulled down	1-2
J90	SYSMON VREFP • 1-2: 1.25V VREFP connected to FPGA • 2-3: VREFP connected to GND	1-2
J20	Reset Sequencer PS_POR_B • OFF: No sequencer control of PS_POR_B • 1-2: Sequencer can control PS_POR_B	1-2
J21	Reset Sequencer PS_SRST_B • OFF: No sequence control of PS_SRST_B • 1-2: Sequencer can control PS_SRST_B	1-2
J22	Reset Sequencer inhibit • OFF: Sequencer normal operation • 1-2: Sequencer inhibit (resets will stay asserted)	OFF
J14	ARM Debug VTREF • Open: VTREF floating • 1-2: VTREF = VCCOPS3 (1.8V)	1-2
J15	ARM Debug VSUPPLY • OFF: VSUPPLY floating • 1-2: VSUPPLY = VCCOPS3 (1.8V)	OFF
J56	VCCO_PSDDR_504 select • 1-2: Switched DDR4 VDDQ • 3-4: Direct DDR4 VDDQ	1-2
J159	DDR4 Reset Suspend Enable • 1-2: Suspend disabled (Gate bypass) • 2-3: Suspend enabled	1-2
J16	SFP0 TX: 1-2:Disable; OFF: Enable	OFF

Jumper	Function	Default
J17	SFP1 TX: 1-2:Disable; OFF: Enable	OFF
J42	SFP2 TX: 1-2:Disable; OFF: Enable	OFF
J54	SFP3 TX: 1-2:Disable; OFF: Enable	OFF
J162	PCIe PRSNT select • 1-2: x1 • 3-4: x4 • 5-6: GND (not used)	5-6
J110	USB ULPI CVBUS Select • 1-2: DEVICE or OTG Mode • 2-3: Host Mode	1-2
J109	USB ULPI ID select • 1-2: Connector ID • 2-3: VDD33 ID	2-3
J112	USB ULPI Shield GND select • 1-2: Capacitor • 2-3: GND	1-2
J7	USB ULPI Device or Host select • 1-2: HOST/OTG • Open: Device	OPEN
J113	USB ULPI Device/Host or OTG select • 1-2: Device or Host • 2-3: OTG	1-2
J88	ARM Trace VTREF • 1-2: 3.3V • Open: 0V	1-2
J38	ARM Trace power • 1-2: 3.3V • Open: 0V	1-2
J153	Power inhibit • OFF: rails power on normally • 1-2: all rails (except UTIL) OFF	OFF
J9	PS_DDR4_VPP_2V5 power inhibit (U39) • OFF: rail powers on normally • 1-2: PS_DDR4_VPP_2V5 OFF	OFF
J164	MSP430 firmware upgrade header	OFF

Hình 3.7. Các Jump cần configure cho ZCU102

Boot Mode	Mode Pins [3:0]	Mode SW6 [4:1]
JTAG	0000	on, on, on, on
QSPI32	0010 <sup>(1)</sup>	on, on, off, on
SD	1110	on, off, on, off

Hình 3.8. Các lựa chọn SW tùy vào Boot Mode

- Ở đây, ta dùng SD card để boot cho mạch, vậy nên sw6 sẽ là 1110.



Hình 3.9. Boot Mode khi sử dụng SD card

- Sau khi điều chỉnh thành công, ta thực hiện bật terminal của ubuntu và kết nối serial máy tính với mạch, sử dụng lệnh dmesg để kiểm tra tên cổng. và thực hiện lệnh như dưới để kết nối

```
tindz@tindz-Lenovo-IdeaPad-S340-15IIL:~$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for tindz:
picocom v3.1

port is      : /dev/ttyUSB0
flowcontrol  : none
baudrate is  : 115200
parity is    : none
databits are : 8
stopbits are : 1
escape is    : C-a
local echo is : no
noinit is    : no
noreset is   : no
hangup is    : no
nolock is    : no
send_cmd is  : SZ -vv
receive_cmd is : RZ -vv -E
imap is      :
omap is      :
emap is      : crclrf,delbs,
logfile is   : none
```

Hình 3.10. Hiện thị picocom trên máy tính

- Sau khi hoàn thành, cuối cùng ta cần bật nguồn cho mạch ZCU102, mạch sẽ tự động bootloader nếu như ta đã làm đúng các bước như trên



```

[ 6.051989] ALSA device list:
[ 6.054952]   #0: DisplayPort monitor
[ 6.058923] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[ 6.067541] cfg80211: failed to load regulatory.db
[ 6.080511] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 6.088632] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[ 6.098816] devtmpfs: mounted
[ 6.101960] Freeing unused kernel memory: 704K
[ 6.113139] Run /sbin/init as init process
INIT: version 2.88 booting
[ 6.273138] [drm] Cannot find any crtc or sizes
Starting udev
[ 6.499326] udevd[168]: starting version 3.2.8
[ 6.509781] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.516972] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.526737] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.561136] udevd[169]: starting eudev-3.2.8
[ 7.058580] cramfs: Unknown parameter 'umask'
[ 7.064535] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[ 7.105522] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
INIT: Entering runlevel: 5
Configuring network interfaces... [ 7.399929] pps pps0: new PPS source ptp0
[ 7.403970] macb ff0e0000.ethernet: gem-ptp-timer ptp clock registered.
udhcpd: started, v1.31.0
udhcpd: sending discover
udhcpd: sending discover
udhcpd: sending discover
udhcpd: no lease, forking to background
done.
Starting haveged: haveged: listening socket at 3
haveged: haveged starting up

Starting Dropbear SSH server: dropbear.
Starting internet superserver: inetd.
Starting syslogd/klogd: done
Starting tcf-agent: [ 17.400862] random: crng init done
[ 17.404276] random: 7 urandom warning(s) missed due to ratelimiting
OK
PetaLinux 2020.1 zcu102_custom_trung /dev/ttyPS0

```

Hình 3.11. Quá trình boot thành công

## 4. Chạy Thread trên KIT ZCU102

### 4.1. Chuẩn bị phần cứng

- Ta cần có bo mạch ZCU102, dây mạng (2 dây, 1 dây nối vào PC, 1 dây nối vào KIT) được kết nối với server, sd card, dây nguồn, dây nối USB đến cổng UART của KIT



Hình 4.1. Chuẩn bị ZCU102 và các thiết bị phù hợp

- Sau khi đã chuẩn bị thành công, thực hiện các bước như bên dưới

#### 4.2. Kiểm tra IP address

```
dell@dell-Precision-7540: ~  
dell@dell-Precision-7540: ~ 80x24  
dell@dell-Precision-7540:~$ sudo picocom -b 115200 /dev/ttyUSB0
```

- Sau khi mở picocom, ta thực hiện bật nguồn cho mạch ZCU102 để boot.
- Sau khi boot thành công, sẽ đăng nhập vào hệ điều hành của mạch với pass là root

```
PetaLinux 2020.1 xilinx-zcu102-2020_1 /dev/ttyPS0  
xilinx-zcu102-2020_1 login: root  
Password:  
root@xilinx-zcu102-2020_1:~#
```

Hình 4.2. Đăng nhập vào hệ điều hành của ZCU102

- Sau khi đăng nhập thành công, tiến hành check IP address của KIT, gõ ip  
a. Ở đây, địa chỉ IP của mạch là 192.168.0.107

### 4.3. SSH máy tính đến Server

- Sau khi đã boot thành công cho mạch, ta cần ssh máy tính đến với IP của server. Ở đây, địa chỉ của server là 192.168.0.236. Sau khi đã ssh tới root(server) thành công, ta thực hiện ssh tới KIT ZCU102 bằng IP của mạch như phần trên

```
dell@dell-Precision-7540:~$ ssh root@192.168.0.236
root@192.168.0.236's password:
Activate the web console with: systemctl enable --now cockpit.socket

Register this system with Red Hat Insights: insights-client --register
Create an account or view all your systems at https://red.ht/insights-dashboard
Last login: Wed Aug 17 14:24:00 2022 from 192.167.120.238
```

```
[root@localhost ~]# ssh root@192.168.0.102
The authenticity of host '192.168.0.102 (192.168.0.102)' can't be established.
RSA key fingerprint is SHA256:6Hj5rXJwTX2Nl7GY5vUFH17XpcP2opchuKYisROQv0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.102' (RSA) to the list of known hosts.
root@192.168.0.102's password:
```

Hình 4.3. Quá trình SSH đến server

### 4.4. Cài đặt và thực hiện cross compile

- Vì mạch ZCU102 sử dụng kiến trúc aarch64, vậy nên ta cần cross compile như sau:
- Thực hiện câu lệnh như sau để install

```
sudo apt install pkg-config-aarch64-linux-gnu
```

- Sau khi install, tiến hành giải nén file và cd vào thư mục bin. Tiếp tục gõ pwd để lấy đường dẫn. Để có thể sử dụng được cross compile aarch64 thực hiện lệnh

```
[root@localhost bin]# export PATH=$PATH:/home/trung/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu/bin
```

- Sau khi thành công, ta cần cross file.c sang aarch64 để chuyển qua mạch ZCU102.

```
[root@localhost BT_Week1]# aarch64-none-linux-gnu-gcc -pthread -o thread Thread.c
```



## 4.5. Chuyển file đã compile sang KIT

- Ta sử dụng câu lệnh SCP như sau

```
root@xilinx-zcu102-2020_1:~# scp thread root@192.168.0.236:/home/linuxize/project/ThucTap_VHT/BT_Week1/

Host '192.168.0.236' is not in the trusted hosts file.
(ecdsa-sha2-nistp256 fingerprint sha1!! b8:f1:02:64:b1:13:ec:bf:db:63:30:ca:d3:62:e2:a0:95:13:01:05)
Do you want to continue connecting? (y/n) y
root@192.168.0.236's password:
thread                                                                100%  19KB  18.5KB/s   00:00
root@xilinx-zcu102-2020_1:~#
```

Hình 4.4. Gửi file đã compile qua Kit ZCU102

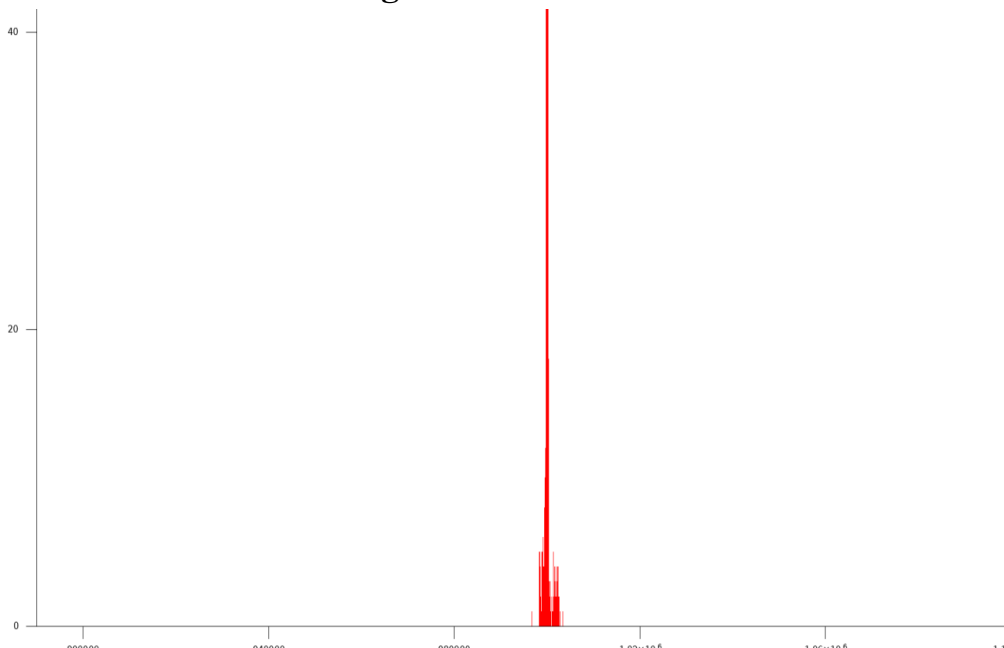
- Sau khi chuyển xong, thực hiện chạy chương trình, gửi lại kết quả thu được về PC bằng cách tương tự.

## 4.6. Kết quả thu được

Ta thực hiện chạy chương trình C trên bo mạch ZCU102 để kiểm tra tính realtime của kernel trên KIT. Từ những kết quả thu được, ta có thể đưa ra được đánh giá về hiệu năng cũng như độ chính xác của CPU có trên KIT. Từ đó, so sánh với hiệu năng trên PC để biết được sự khác nhau giữa tính realtime trên KIT và PC là như thế nào

### 4.6.1. Khi chạy độc lập

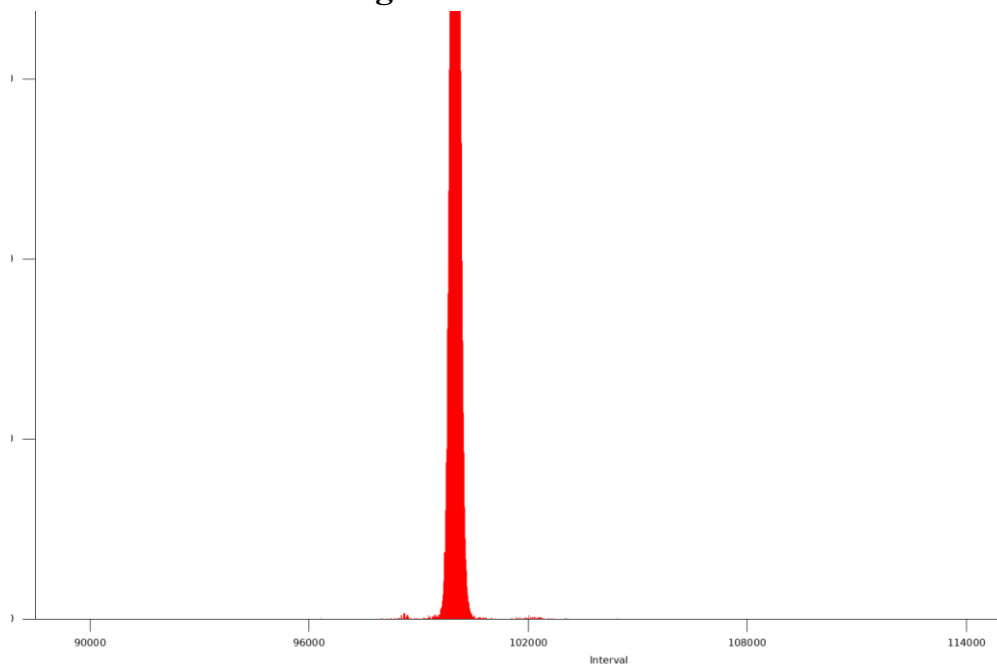
- **Khảo sát khi chu kì bằng 1000000ns**



Hình 4.5. Đồ thị phân bố của biến khi T = 1000000ns

Nhận xét: Dễ dàng nhận thấy, với  $T = 1000000\text{ns}$  thì giá trị offset ta thu được phần lớn, đều nằm trong khoảng  $1000000\text{ ns}$ . Điều này tương tự khi ta chạy trên PC

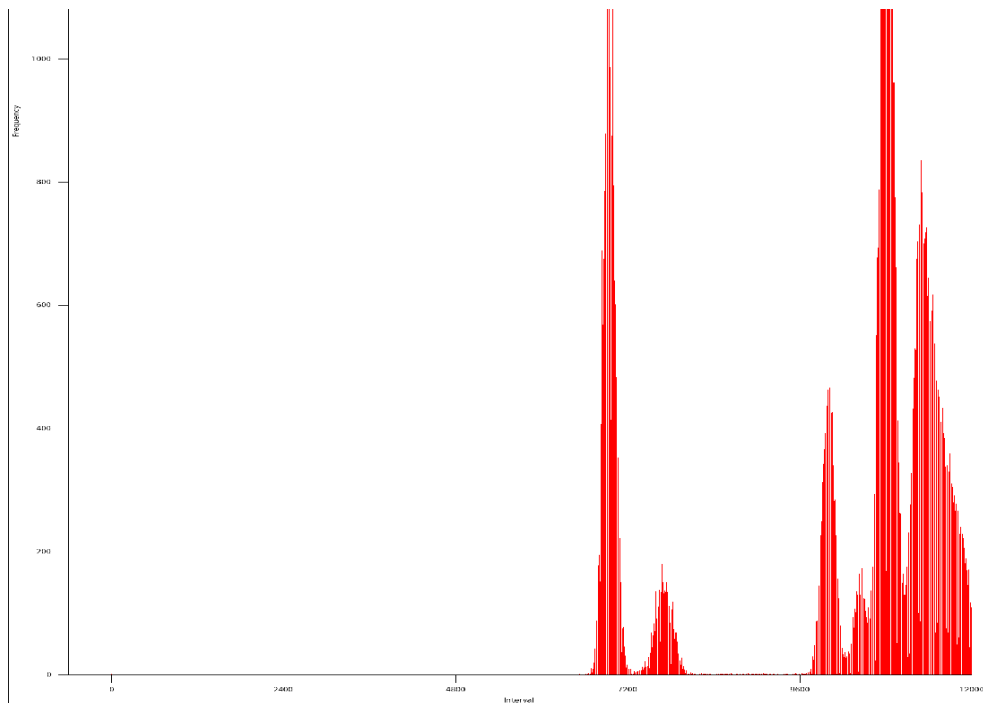
- **Khảo sát khi chu kì bằng  $100000\text{ns}$**



Hình 4.6. Đồ thị phân bố của biến khi  $T = 100000\text{ns}$

Nhận xét: Với giá trị  $100000\text{ns}$ , offset cho ra không quá khác biệt khi ta sử dụng PC, có thể thấy vùng giá trị đã dày lên, tuy nhiên luôn giao động quanh mức  $100000\text{ns}$

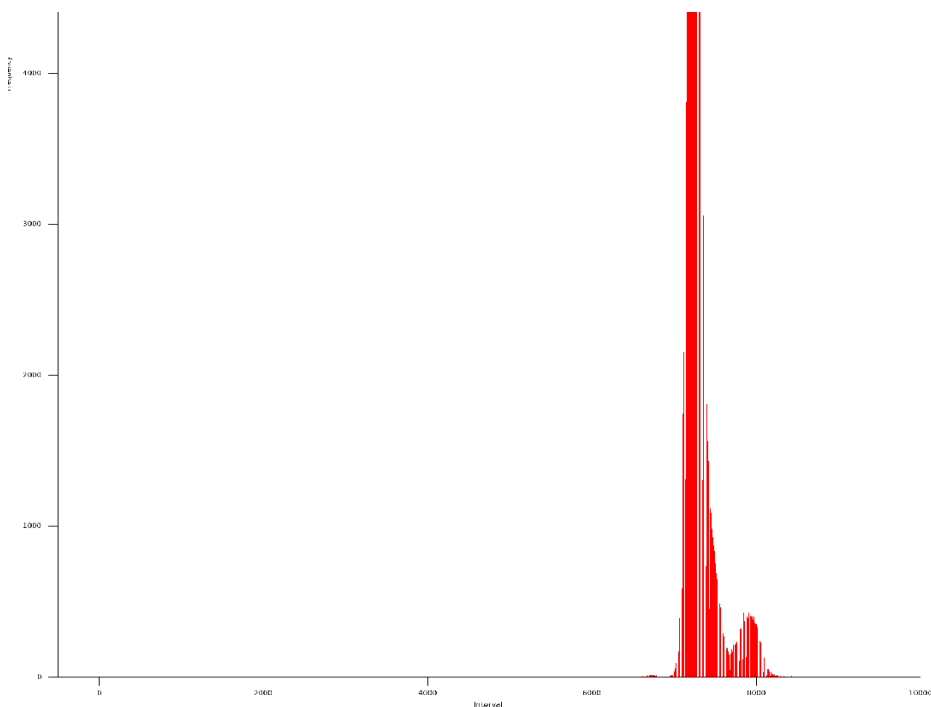
- **Khảo sát khi chu kì bằng  $10000\text{ns}$**



Hình 4.7. Đồ thị phân bố của biến khi  $T = 10000\text{ns}$

Nhận xét: Lúc này ta bắt đầu thấy có sự khác biệt, vùng giá trị  $10000\text{ns}$  vẫn tương đối dày với lượng giá trị lớn, tuy nhiên đã xuất hiện biến số khi có thêm 1 vùng giá trị  $7000\text{ns}$ . Điều này là điểm khác so với PC.

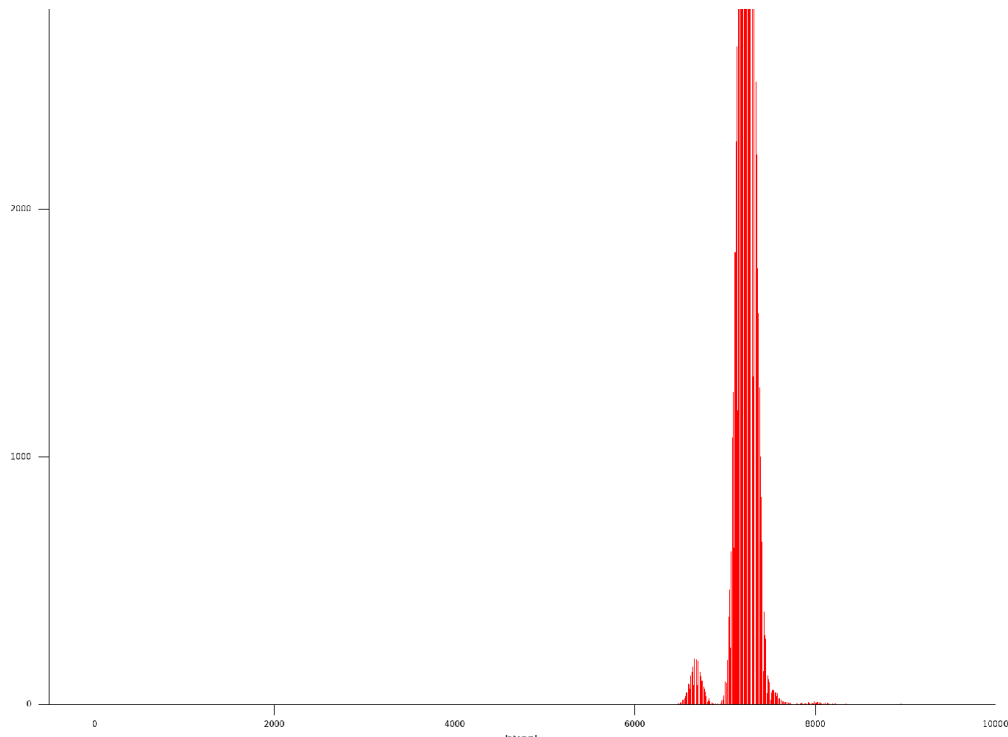
#### - Khảo sát khi chu kì là $1000\text{ns}$



Hình 4.8. Đồ thị phân bố của biến khi  $T = 1000\text{ns}$

Nhận xét: Đến vùng giá trị 1000, offset cho giao động trong khoảng 6000 - 8000. Điều này cho thấy, đây có thể đã là giới hạn của KIT, và nó cũng đồng thời tương đồng với giá trị khi chúng ta sử dụng PC, tuy nhiên đối với PC ngoài khoảng giá giá 6000-8000 thì còn một khoảng giá trị từ 2000-4000. Điều này cho thấy CPU của PC có thể xử lý tốt hơn của KIT

- **Khảo sát khi chu kì là 100ns**



Hình 4.9. Đồ thị phân bố của biến khi  $T = 100\text{ns}$

Nhận xét: Đối với  $T = 100\text{ns}$ , cũng tương tự như giá trị 1000 khi mà offset đã tới giới hạn thì việc chúng ta thay đổi freq xuống là vô nghĩa. So sánh với giá trị khi thực hiện trên PC thì kết quả không chính xác bằng

#### 4.6.2. Kết luận – đánh giá

- Sau khi khảo sát các giá trị lấy mẫu của biến thông qua việc thay đổi chu kì để đưa ra được những đánh giá về CPU của KIT, từ đó có thể so sánh khi ta thực hiện chạy trên mạch. Có một vài quan điểm như sau:
- Với giá trị chu kì bằng 1000000ns, 100000ns, 10000ns, thông qua đồ thị biến và đồ thị phân bố của biến, dễ dàng nhận thấy độ ổn định của việc lấy mẫu là rất tốt, khi giá trị lấy mẫu trả về luôn phân bố xung quanh giá trị chu kì. Qua đó cho thấy rằng CPU hoạt động ổn định và không xảy ra

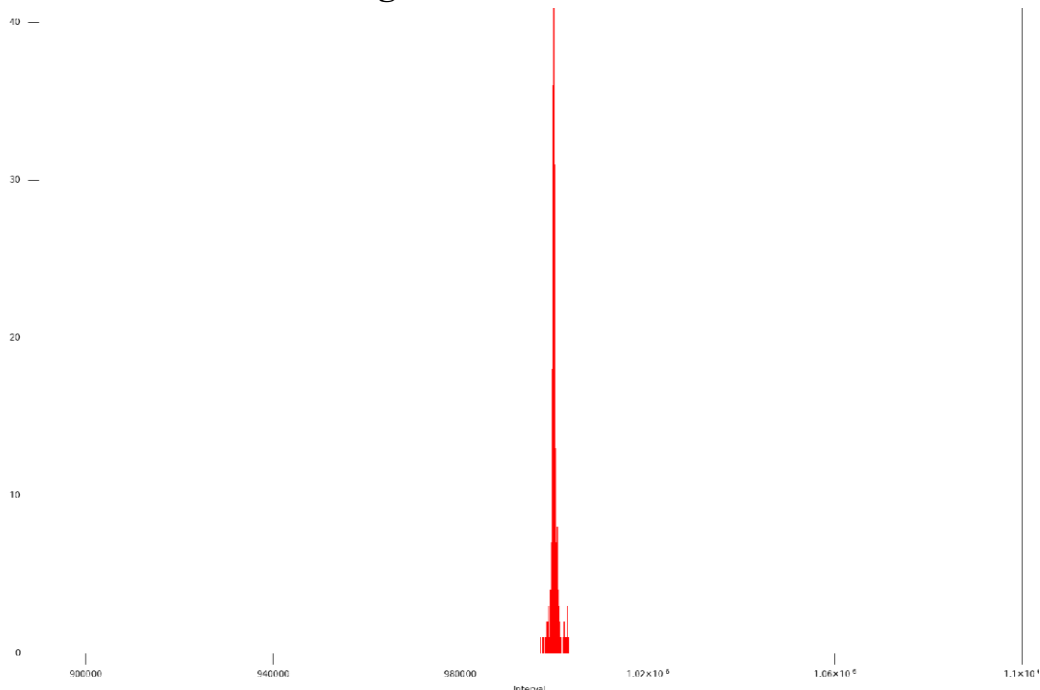
biến số nhiều khi T ở khoảng cao, điều này tương tự đối khi ta chạy trên PC

- Tuy nhiên, khi ta lấy mẫu với giá trị chu kì bằng 1000ns, 100ns, thu được kết quả đã có sự khác nhau. Trong khi PC có thể thu được giá trị lấy mẫu giao động trong khoảng [2500ns-3000ns] thì trên KIT ta chỉ có thể thu được giá trị giao động trong khoảng [6000ns-8000ns]. Từ đó ta có thể đưa ra kết luận. Hiệu năng của cả CPU trên cả KIT và PC đều tương đối tốt, tuy nhiên với PC hiệu năng tốt hơn so với KIT.

#### 4.6.3. Khi chạy có tải( file disturb)

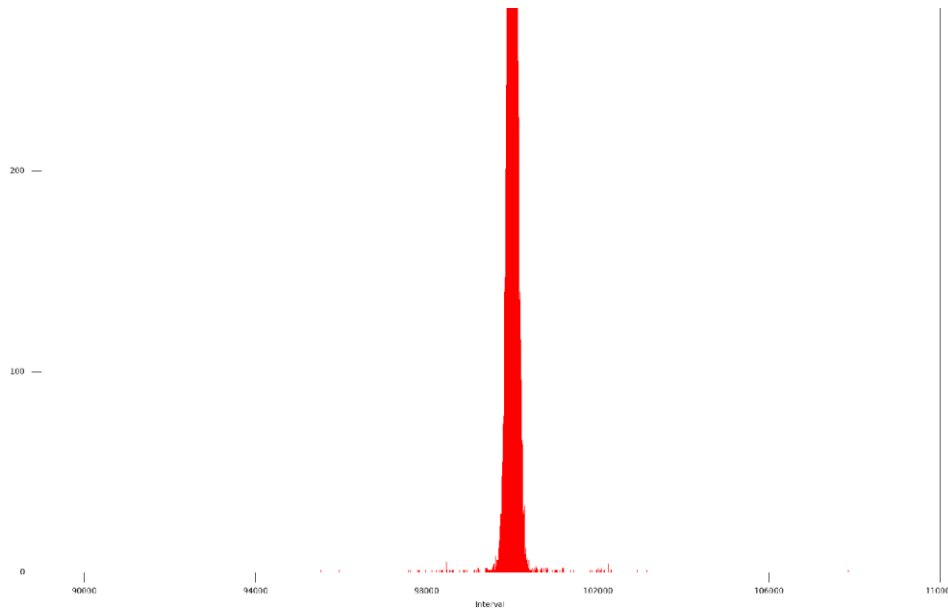
- Để tiếp tục kiểm tra hiệu năng của CPU, ta cần tiến hành 1 bài test, ta sẽ chạy 1 tiến trình disturb với mục đích chiếm dụng tài nguyên của CPU. Sau đó sẽ thực hiện tiến trình C, từ kết quả ta sẽ so sánh với khi không chạy tiến trình disturb
- Khi ta thực hiện chạy file disturb, điều này sẽ làm cho CPU phải làm thêm việc, nói cách khác nó đang bị làm phiền và phải thực hiện cả code mà chúng ta đang muốn chạy cả file disturb
- Ta có thể sử dụng lệnh top trên terminal để kiểm tra % CPU bị chiếm dụng

#### - Khảo sát khi chu kì bằng 1000000ns



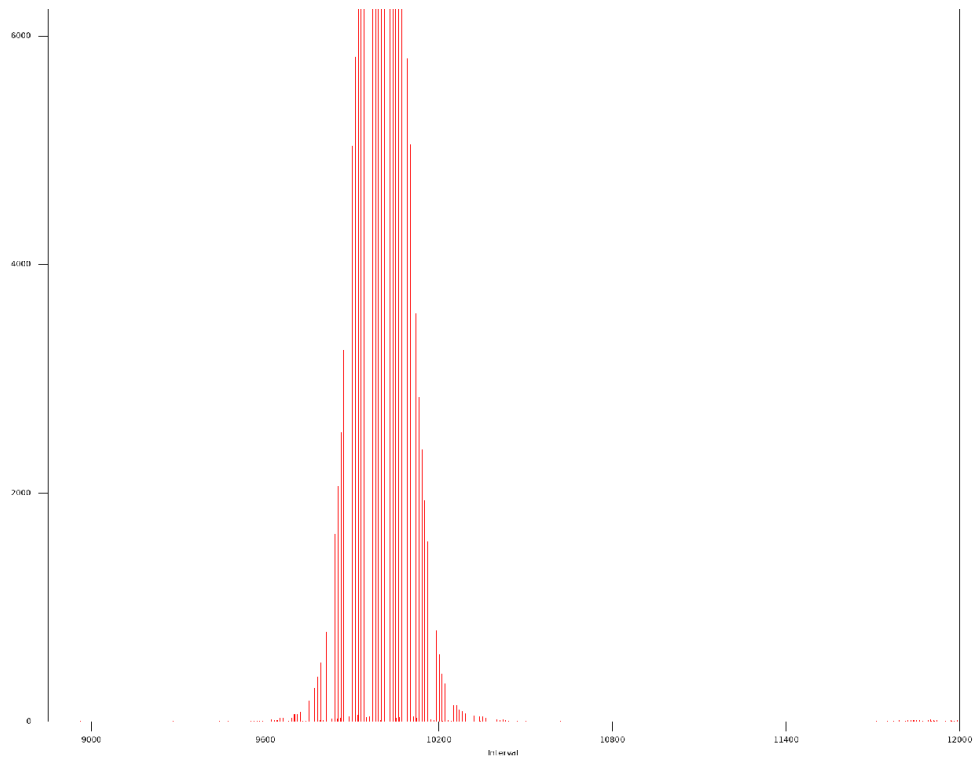
Hình 4.10. Đồ thị biến khi T = 1000000ns ( có disturb)

- **Khảo sát khi chu kì 100000ns**



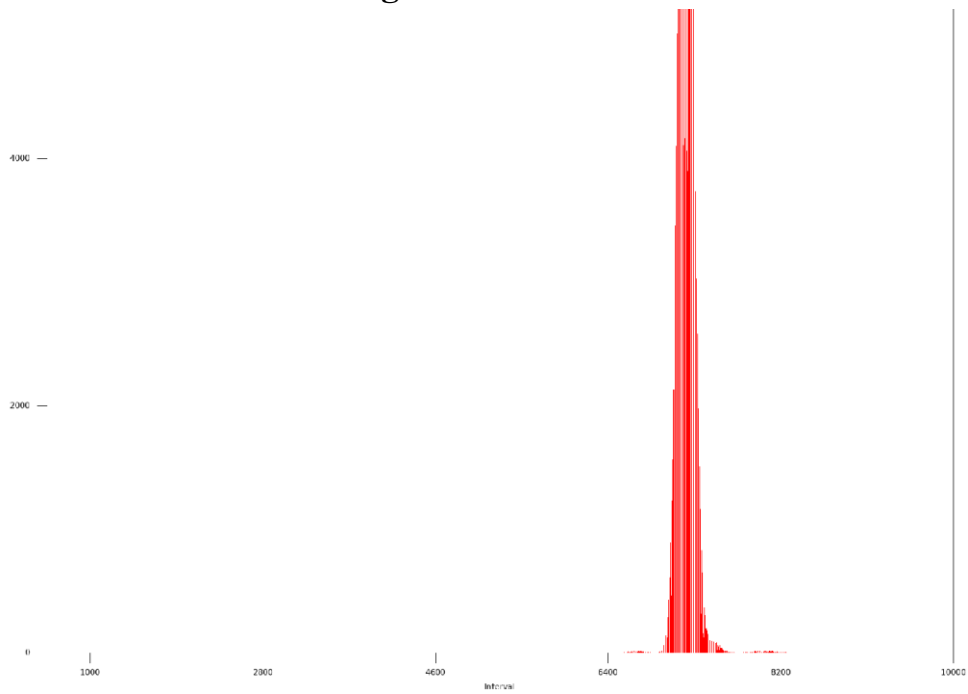
Hình 4.11. Đồ thị phân bố của biến khi  $T = 100000\text{ns}$  ( có disturb)

- **Khảo sát khi chu kì bằng 10000ns**



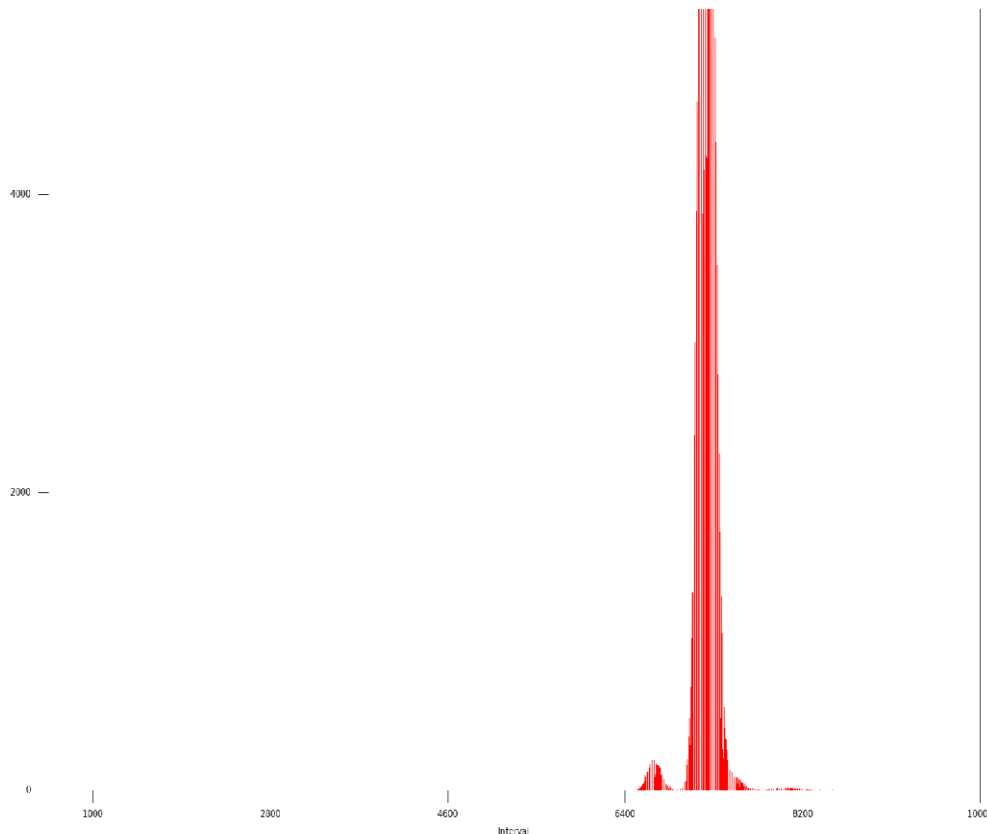
Hình 4.11. Đồ thị phân bố của biến khi  $T = 10000\text{ns}$  ( có disturb)

- **Khảo sát khi chu kì bằng 1000ns**



Hình 4.12. Đồ thị phân bố của biến khi  $T = 1000\text{ns}$  ( có disturb)

- **Khảo sát khi chu kì bằng 100ns**



Hình 4.13. Đồ thị phân bố của biến khi  $T = 100\text{ns}$  ( có disturb)

#### 4.6.4. Kết luận – đánh giá

- Ta cần đánh giá sự khác nhau, cụ thể là hiệu năng của CPU trên KIT, khi ta thực hiện chạy 1 file ẩn để chiếm dụng nguồn tài nguyên của CPU, so với khi nó chạy độc lập
- Có thể khi ta chạy thread SAMPLE cùng với disturb. Hệ thống sẽ tự động phân cấp các tiến trình tùy vào độ quan trọng của nó. Độ ưu tiên của tiến trình: Các tiến trình có thể được phân cấp theo một số tiêu chuẩn đánh giá nào đó, một cách hợp lý, các tiến trình quan trọng hơn (có độ ưu tiên cao hơn) cần được ưu tiên cao hơn. Cụ thể, với những tiến trình phức tạp hơn, sử dụng nhiều hàm và có nhiều tiền tố hơn thông thường sẽ có độ ưu tiên cao hơn.
- Nhìn chung, gần như các giá trị khi chúng ta chạy chương trình cùng file disturb đều không quá bị ảnh hưởng so với khi chạy độc lập, điều đó có thể phần nào cho thấy, độ ưu tiên của tiến trình disturb thấp hơn so với thread, mặc dù nó có chiếm dụng CPU, từ đó ta có thể kết luận rằng độ ưu tiên cũng rất quan trọng khi ta chạy đồng thời nhiều tiến trình khác nhau.



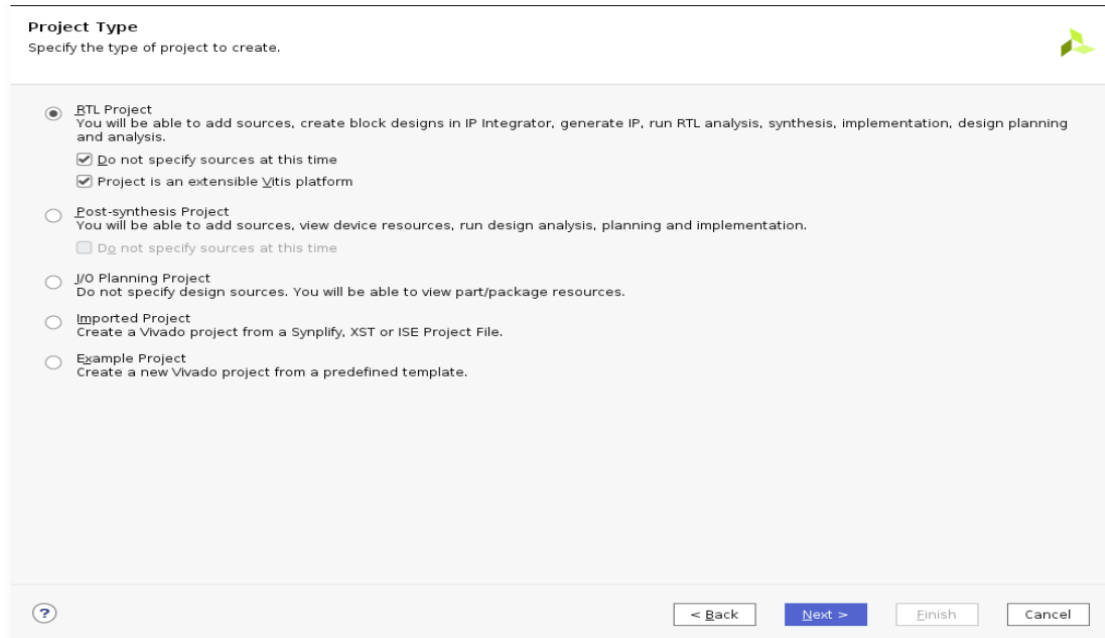
## 5. Xây dựng images từ file Hardware tự Design cho KIT ZCU102

### 5.1. Sử dụng vivado để design IP

- Thực hiện chạy câu lệnh để cài đặt môi trường cho vitis\_HLS và vivado

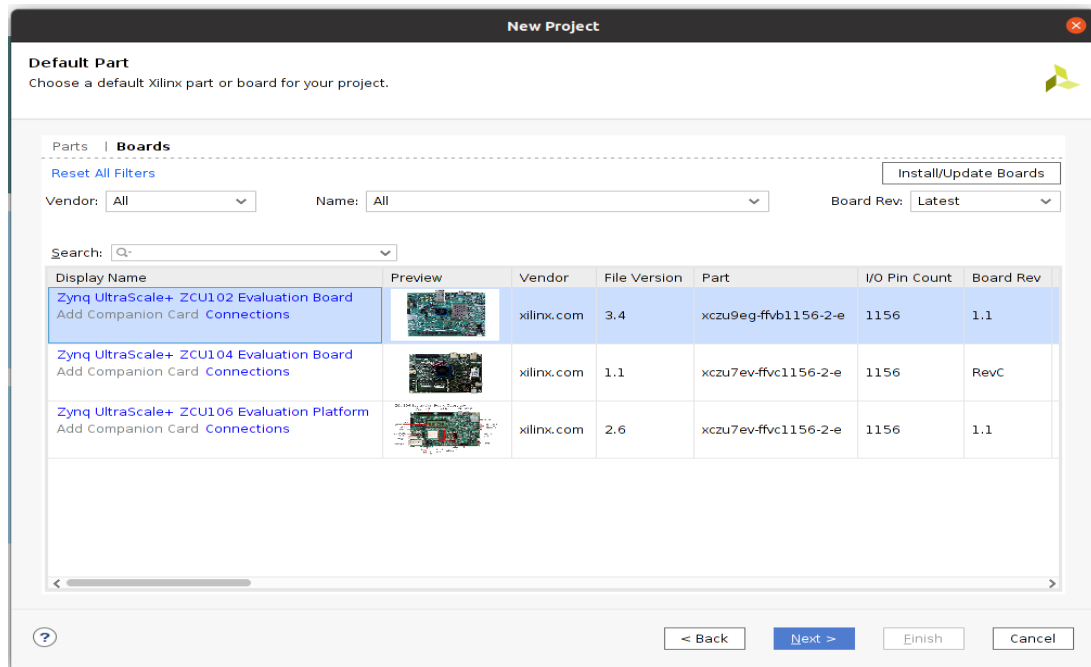
```
source <Vitis_Install_Directory>/settings64.sh  
vivado &
```

- Chọn File, tiếp tục chọn Project, chọn New, Click Next.



Hình 5.1. Lựa chọn kiểu Project phù hợp

- Chọn KIT ZCU102, chọn next



Hình 5.2. Lựa chọn KIT phù hợp

- Tiến hành click vào Create Block Design để thiết kế

▼ IP INTEGRATOR

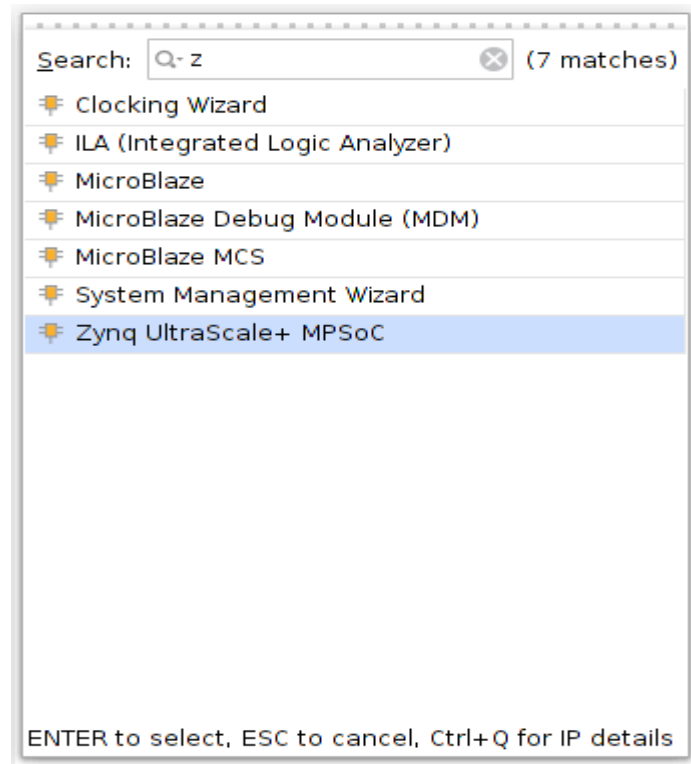
[Create Block Design](#)

[Open Block Design](#)

[Generate Block Design](#)

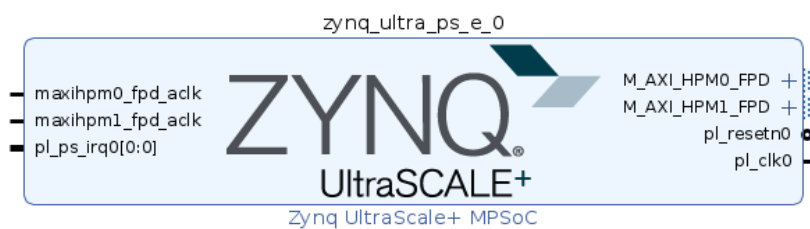
[Export Platform](#)

- Chọn Add IP, tìm kiếm zynq UltraScale+ MPSoC



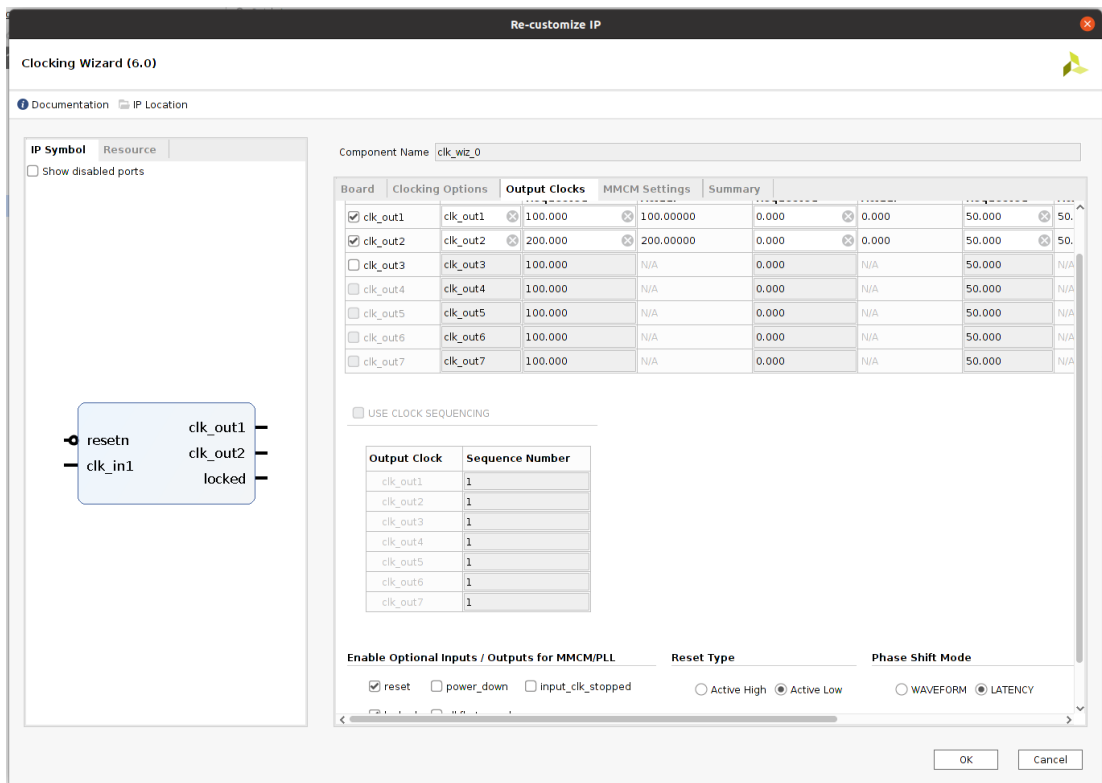
Hình 5.3. Tìm kiếm Zynq UltraScale+ MPSoc

- Sau khi click vào, ta thực hiện Run connection automatic



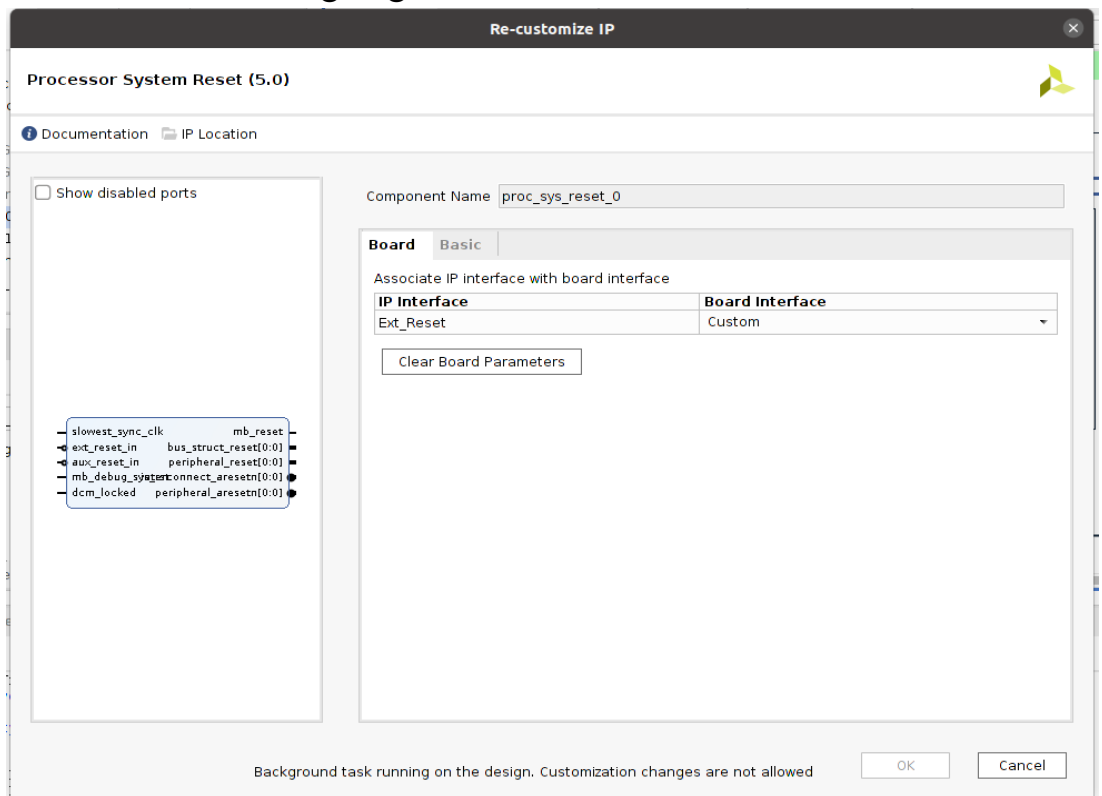
Hình 5.4. Zynq UltraScale

- Tiếp theo, chúng ta cần custom cho clock. Ta tiến hành Add IP clock wizard vào thiết kế. Thực hiện configuration như bên dưới, chọn 2 clock và đổi reset type sang Low.



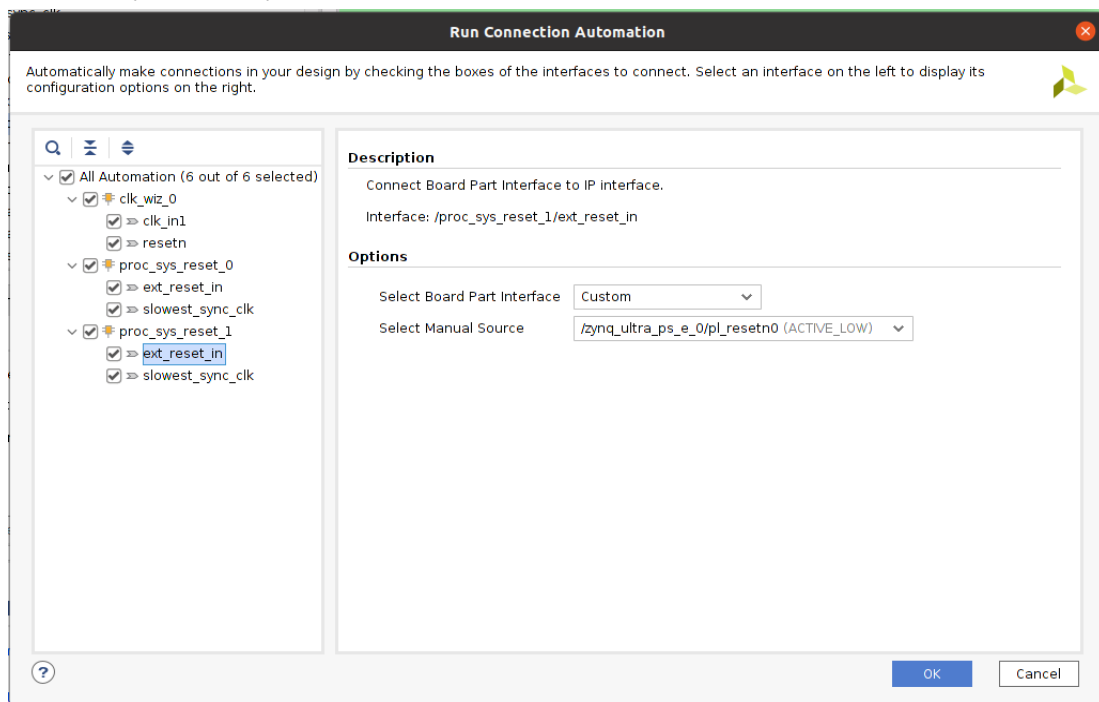
Hình 5.5. Configure clock wizard

- Sau khi đã configure clock xong, ta tiếp tục chọn Add IP reset, ở đây ta cần 2 khối reset tương ứng với 2 clock



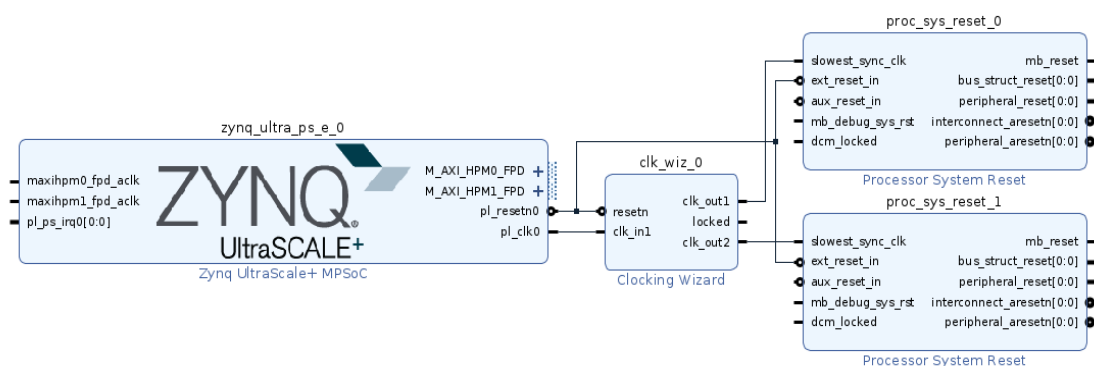
Hình 5.6. Configure Processor System Reset

- Sau khi hoàn thành, ta run automatic. Thực hiện thay đổi giá trị reset từ auto sang active\_low. Thay đổi clock của từng khối reset theo clock ban đầu đã tạo lần lượt là 100000 và 200000



Hình 5.7. Configire khi run automatic

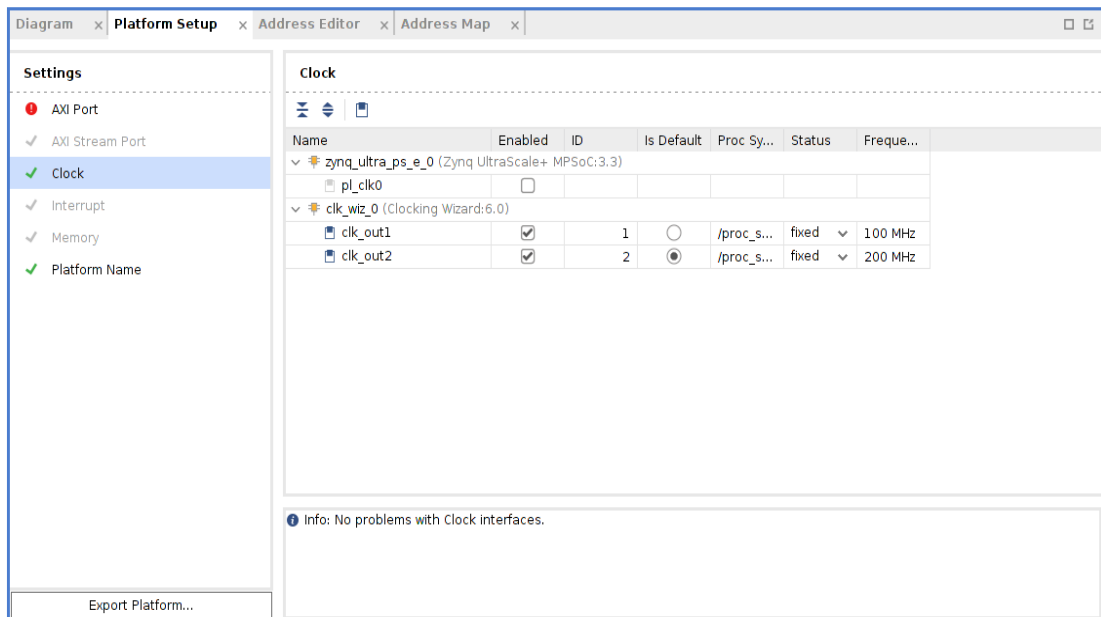
- Sau khi hoàn thành, ta sẽ được thiết kế như hình dưới



### Hình 5.8. Thiết kế sau khi hoàn thành

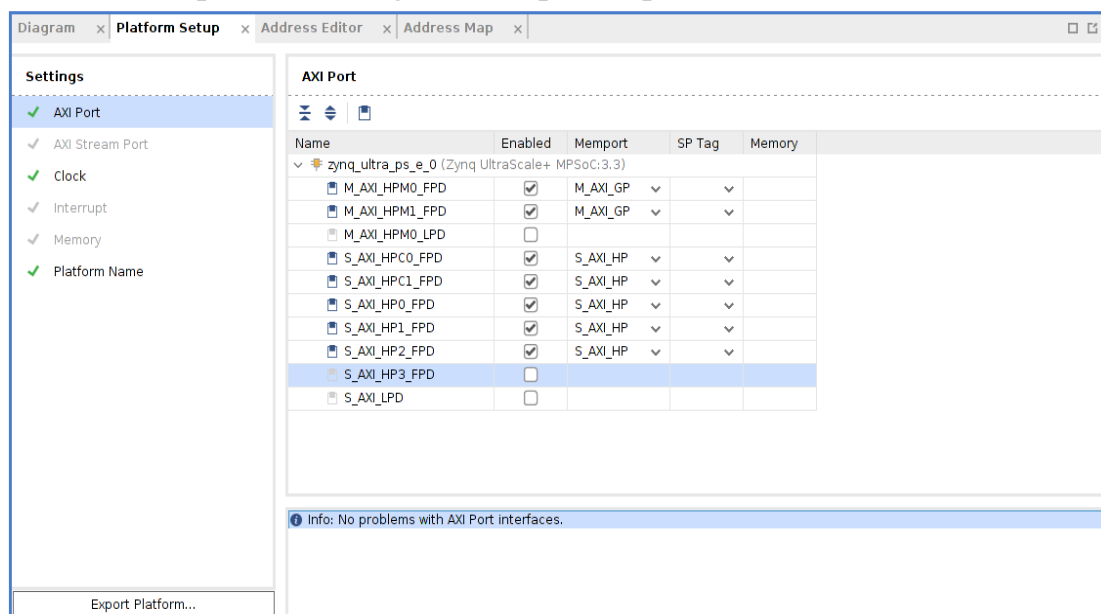
- Tiếp theo, chúng ta cần setup platform cho thiết kế. Thực hiện chọn window, chọn setup platform. Lần lượt điều chỉnh clock và AXI như hình dưới đây.

- Ở mục clock, ta cần cho phép clock đã tạo hoạt động, và chọn 1 clock làm mặc định



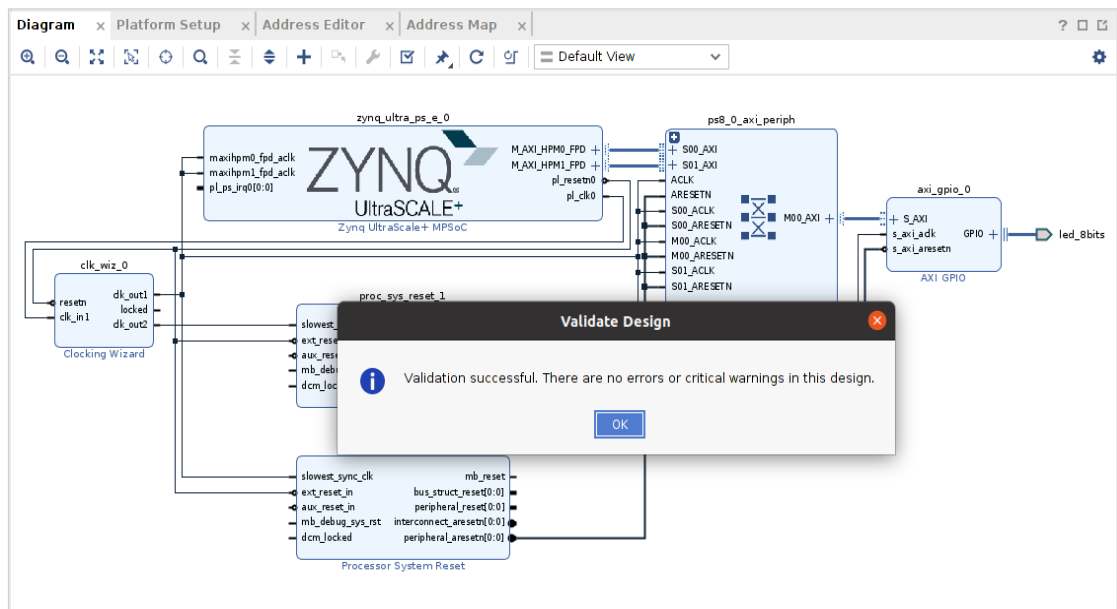
Hình 5.9. Chỉnh sửa clock trong platform setup

- Trong khi đó, đối với AXI, ta thực hiện cho phép các AXI hoạt động, thay đổi các memport và SP tag sao cho phù hợp.



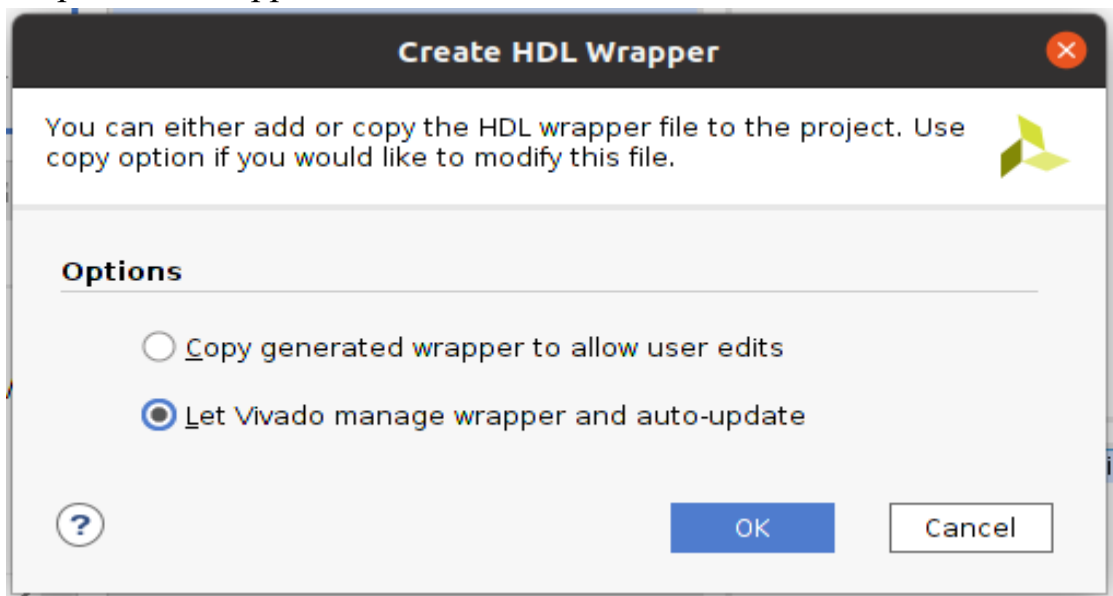
Hình 5.10. Chỉnh sửa AXI Port trong Platform setup

- Sau khi đã hoàn thành tất cả, ta thực hiện validate thiết kế.



Hình 5.11. Validate thiết kế

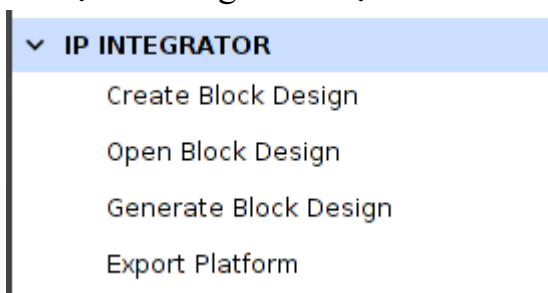
- Tiếp tục tạo wrapper cho thiết kế



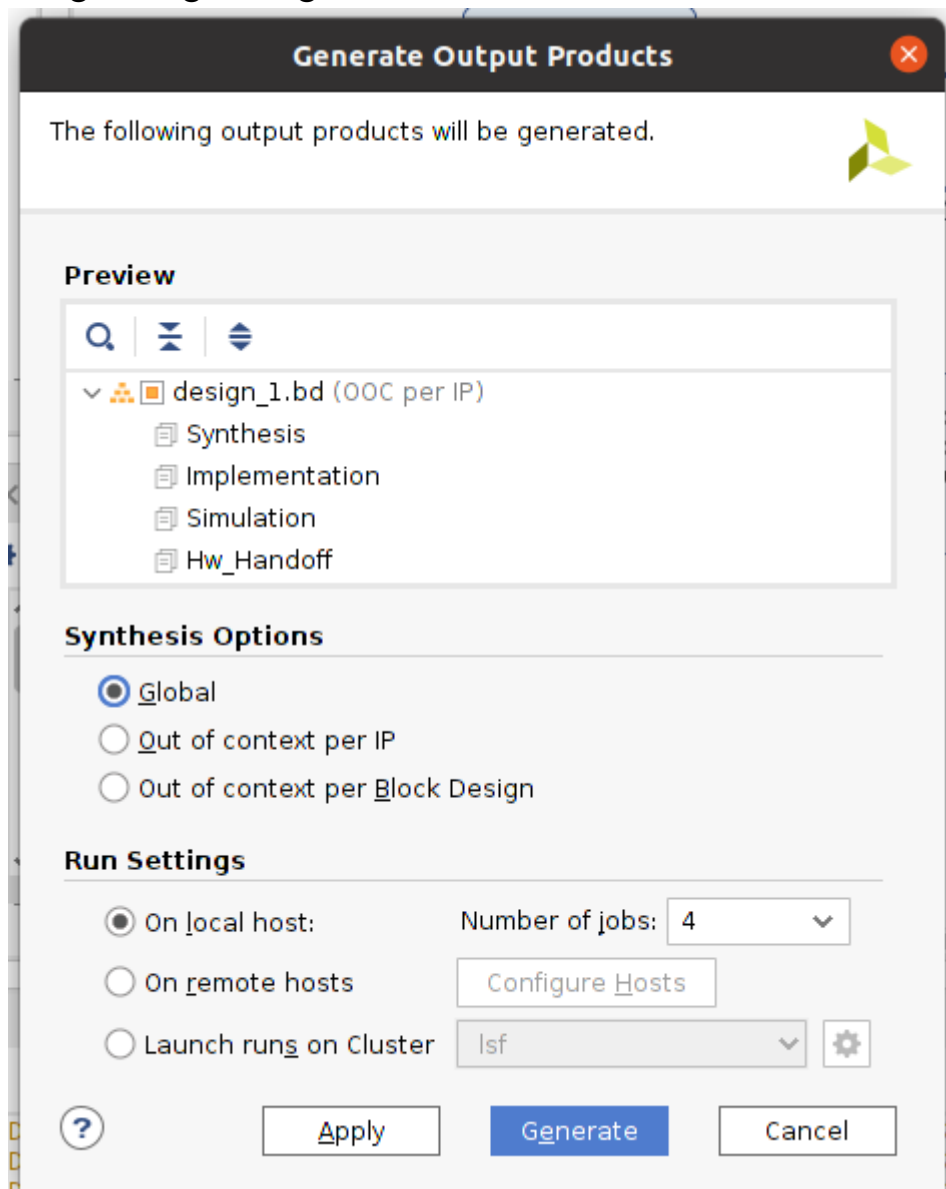
Hình 5.12. Tạo wrapper cho thiết kế

## 5.2. Xuất ra file Hardware

- Ở mục IP Integrator chọn Generate block design



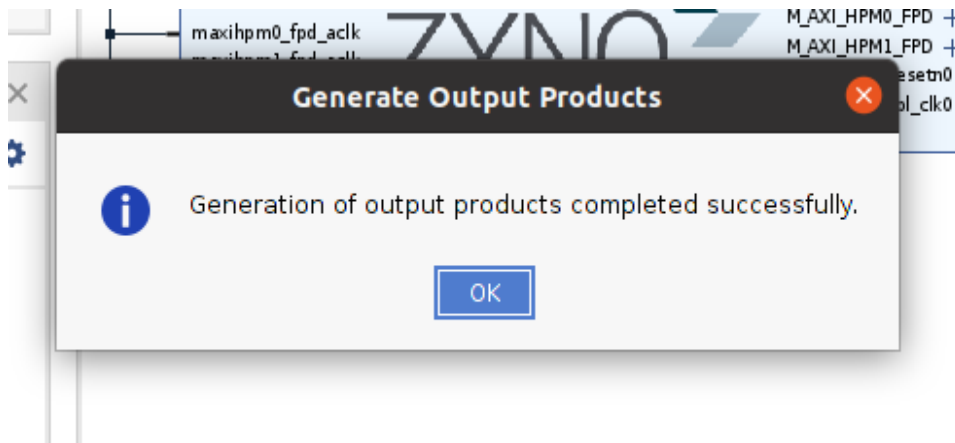
- Ta chọn local để ko cần tổng hợp file XDA, mà chỉ thực hiện xuất ra với design thông thường



Hình 5.13. Generate Block Design

- Thành công





Hình 5.14. Generate thành công

- Tiếp theo ta cần chọn generate Bitstream

▼ PROGRAM AND DEBUG

Generate Bitstream

▼ Open Hardware Manager

- Sau khi hoàn thành các bước, ta cần thay đổi PS từ rtl sang tlm

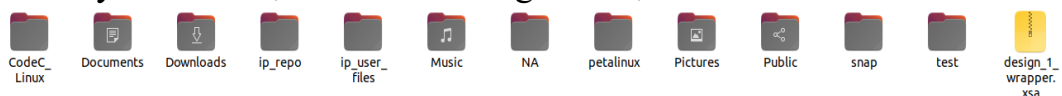
```
set_property SELECTED_SIM_MODEL tlm [get_bd_cells /zynq_ultra_ps_e_0]
```

- Thực hiện những câu lệnh dưới đây trong TCL của vivado để tạo ra file xsa

```
# Setting platform properties
set_property platform.default_output_type "sd_card" [current_project]
set_property platform.design_intent.embedded "true" [current_project]
set_property platform.design_intent.server_managed "false" [current_project]
set_property platform.design_intent.external_host "false" [current_project]
set_property platform.design_intent.datacenter "false" [current_project]
# Write pre-synthesis expandable XSA
write_hw_platform -hw -force -file ./zcu104_custom_platform_hw.xsa
write_hw_platform -hw_emu -force -file ./zcu104_custom_platform_hwemu.xsa
```

Hình 5.15. Setup platform properties và generate XSA file

- Lúc này sẽ xuất hiện file XSA trong thư mục



Hình 5.16. XSA file trong thư mục

### 5.3. Config cho project

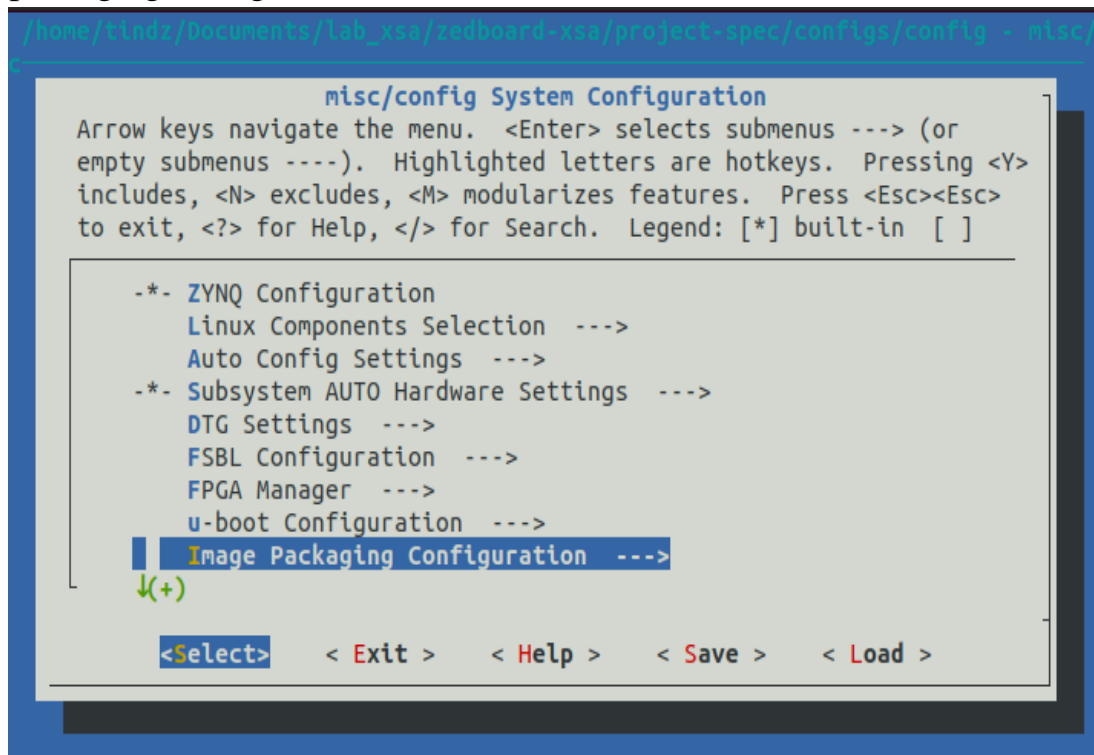
- Cài đặt môi trường cho petalinux

```
source <petalinux_tool_install_dir>/settings.sh
```

- Tạo project theo template và config theo file xsa đã generate từ trước đó

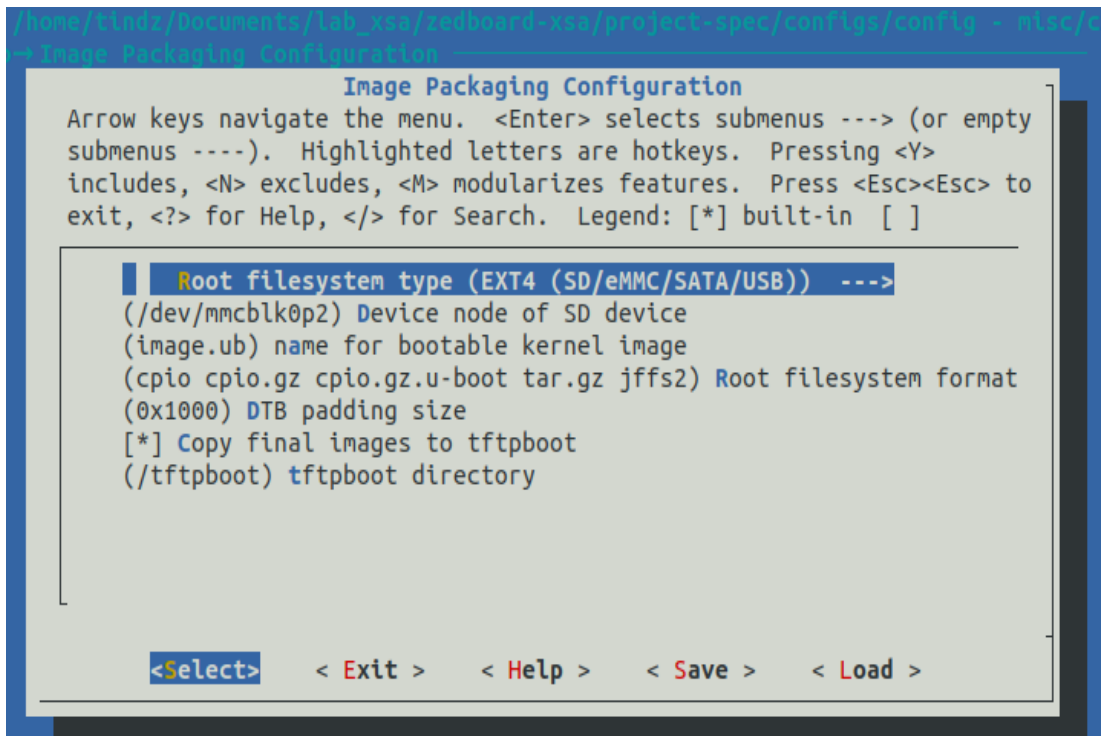
```
petalinux-create --type project --template zynqMP --name zcu104_custom_plnx  
cd zcu104_custom_plnx  
petalinux-config --get-hw-description=<vivado_design_dir> # The directory where your *.xsa
```

- Thực hiện config cho project, sử dụng lệnh petalinux-config, chọn image packaging configuration



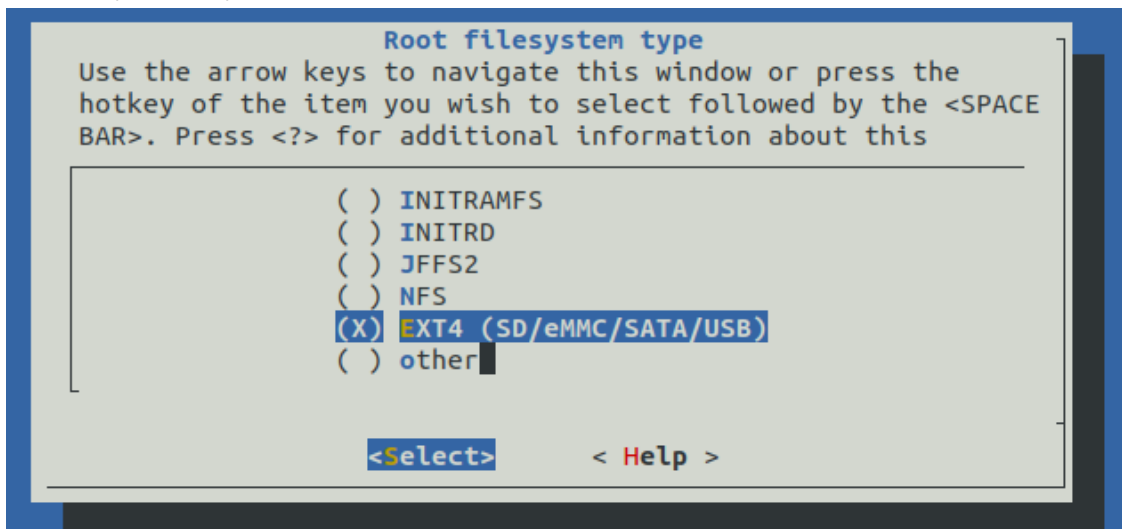
Hình 5.17. Thực hiện configure cho Images

- Tiếp tục chọn rootfs



Hình 5.18. Configure cho Root filesystem

- Lúc này ta thấy có 5 lựa chọn



Hình 5.19. Lựa chọn EXT4

- Nếu ta chọn INITRAMFS thì khi boot cho mạch, kernel sẽ tìm đến ram để tìm rootfs
- Nếu ta chọn EXT4 kernel sẽ tìm đến phân vùng 2 của SD card để tìm rootfs

## 5.4. Build image và boot cho ZCU102

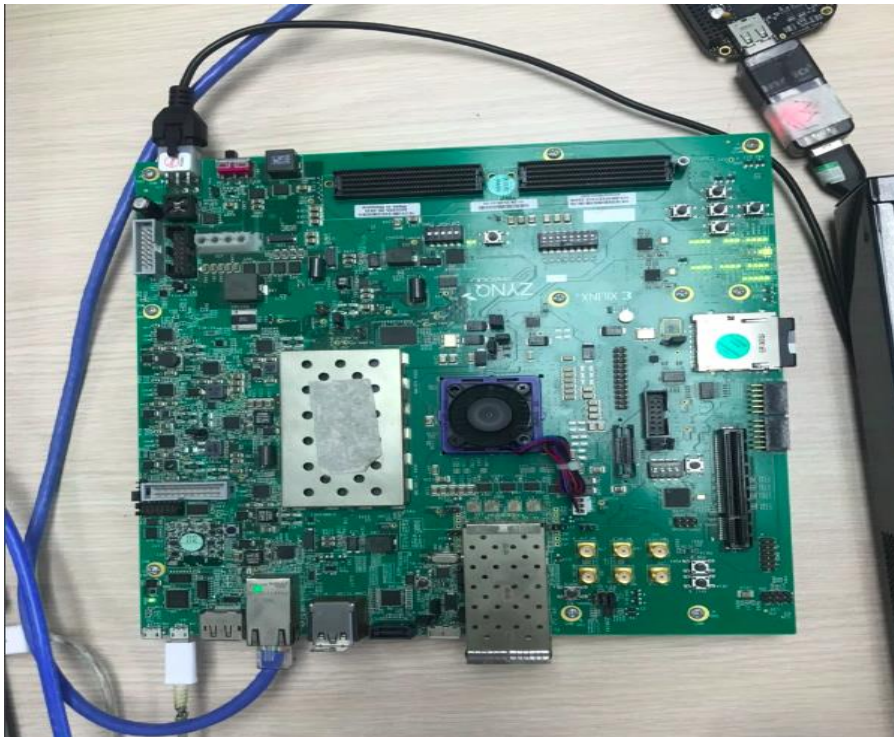
- Khi đã configure xong, ta tiến hành build image

```
[truong@SRV236-REDHAT zcu102_custom_trung]$ petalinux-build
INFO: sourcing build tools
[INFO] building project
[INFO] sourcing build environment
[INFO] generating user layers
[INFO] generating workspace directory
INFO: bitbake petalinux-image-minimal
WARNING: Host distribution "rhel-8.6" has not been validated with this version of the build system; you may possibly experience unexpected failures. It is recommended that you use a tested distribution.
Loading cache: 100% |#####| Time: 0:00:00
Loaded 4229 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2961 .bb files complete (2960 cached, 1 parsed). 4230 targets, 168 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####| Time: 0:00:02
Checking sstate mirror object availability: 100% |#####| Time: 0:00:02
Sstate summary: Wanted 128 Found 14 Missed 114 Current 888 (10% match, 88% complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: Tasks Summary: Attempted 3614 tasks of which 3610 didn't need to be rerun and all succeeded.

Summary: There was 1 WARNING message shown.
INFO: Failed to copy built images to tftp dir: /tftpboot
[INFO] successfully built project
```

Hình 5.20. Tiến hành build images

- Sau khi build thành công, ta cần cắm nguồn mạch, dây USB và SD card như hình dưới đây



Hình 5.21. Chuẩn bị phần cứng

- Tiếp theo, thực hiện mở picocom

```
dell@dell-Precision-7540:~$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for dell:
picocom v3.1

port is      : /dev/ttyUSB0
flowcontrol  : none
baudrate is  : 115200
parity is    : none
databits are : 8
stopbits are : 1
escape is    : C-a
local echo is : no
noinit is    : no
noreset is   : no
hangup is    : no
nolock is    : no
send_cmd is  : SZ -vv
receive_cmd is : RZ -vv -E
imap is      :
omap is      :
emap is      : crcrlf,delbs,
logfile is   : none
initstring   : none
exit_after is : not set
exit is      : no

Type [C-a] [C-h] to see available commands
Terminal ready
█
```

Hình 5.22. Tiến hành bật Picocom

- Tiến hành bật nguồn, lúc này mạch sẽ tự động được boot lên
- Đối với options INITRAMFS: sau khi load xong kernel từ sd card ( file image.ub). Second stage bootloader sẽ chuyển lại quyền điều khiển cho kernel, lúc này kernel sẽ tìm đến RAM để lấy root file system để thực hiện nốt công việc. Một điểm lưu ý khi ta sử dụng initramfs là do file ở trong RAM vậy nên mỗi khi ta reboot thì dữ liệu sẽ mất đi

```

[ 6.468796] ALSA device list:
[ 6.471757] #0: DisplayPort monitor
[ 6.475736] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[ 6.475921] Freeing unused kernel memory: 704K
[ 6.484351] cfg80211: failed to load regulatory.db
[ 6.521194] Run /init as init process
INIT: version 2.88 booting
Starting udev
[ 6.622147] udevd[167]: starting version 3.2.8
[ 6.626928] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.633424] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.639929] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.650842] udevd[168]: starting eudev-3.2.8
[ 6.653215] [drm] Cannot find any crtc or sizes
[ 7.096345] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 7.097000] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
Configuring packages on first boot...
(This may take several minutes. Please do not power off the machine.)
Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing)
INIT: Entering runlevel: 5
Configuring network interfaces... [ 7.467855] pps pps0: new PPS source ptp0
[ 7.471890] macb ff0e0000.ethernet: gem-ptp-timer ptp clock registered.
udhcpc: started, v1.31.0
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: no lease, forking to background
done.
Starting haveged: haveged: listening socket at 3
haveged: haveged starting up

Starting Dropbear SSH server: Generating 2048 bit rsa key, this may take a while...
haveged: haveged: ver: 1.9.5; arch: generic; vend: ; build: (gcc 9.2.0 CTV); collect: 128K

haveged: haveged: cpu: (VC); data: 16K (D); inst: 16K (D); idx: 11/40; sz: 15456/64452

haveged: haveged: tot tests(BA8): A:1/1 B:1/1 continuous tests(B): last entropy estimate 7.99653

haveged: haveged: fills: 0, generated: 0

[ 17.439730] random: crng init done
[ 17.443146] random: 7 urandom warning(s) missed due to ratelimiting

```

```

Petalinux 2020.1 zcu102_custom_trung /dev/ttyPS0

zcu102_custom_trung login: root
Password:
ls -lh

Login incorrect
zcu102_custom_trung login: root
Password:
root@zcu102_custom_trung:~# ls -lh
total 0
root@zcu102_custom_trung:~# mkdir test1 test2 test3
root@zcu102_custom_trung:~# ls -lh
total 0
drwxr-xr-x  2 root    root          0 Aug 31 02:19 test1
drwxr-xr-x  2 root    root          0 Aug 31 02:19 test2
drwxr-xr-x  2 root    root          0 Aug 31 02:19 test3
root@zcu102_custom_trung:~# reboot

```

Hình 5.23. Boot Images thành công khi sử dụng filesystem trong RAM

- Đối với options EXT4: kernel sẽ tìm đến phân vùng rootfs của SD Card để tìm root file system, lúc này ta cần extract file rootfs.cpio bên trong image vào phân vùng rootfs của sd card. Riêng với option EXT4, dữ liệu khi ta thao tác sẽ không bị mất đi như INITRAMFS, thay vào đó nó được lưu lại bên trong SD card



```

[ 6.051989] ALSA device list:
[ 6.054952]   #0: DisplayPort monitor
[ 6.058923] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[ 6.067541] cfg80211: failed to load regulatory.db
[ 6.080511] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 6.088632] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[ 6.098816] devtmpfs: mounted
[ 6.101960] Freeing unused kernel memory: 704K
[ 6.113139] Run /sbin/init as init process
INIT: version 2.88 booting
[ 6.273138] [drm] Cannot find any crtc or sizes
Starting udev
[ 6.499326] udevd[168]: starting version 3.2.8
[ 6.509781] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.516972] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.526737] random: udevd: uninitialized urandom read (16 bytes read)
[ 6.561136] udevd[169]: starting eudev-3.2.8
[ 7.058580] cramfs: Unknown parameter 'umask'
[ 7.064535] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[ 7.105522] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
INIT: Entering runlevel: 5
Configuring network interfaces... [ 7.399929] pps pps0: new PPS source ptp0
[ 7.403970] macb ff0e0000.ethernet: gem-ptp-timer ptp clock registered.
udhcpc: started, v1.31.0
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: no lease, forking to background
done.
Starting haveged: haveged: listening socket at 3
haveged: haveged starting up

Starting Dropbear SSH server: dropbear.
Starting internet superserver: inetd.
Starting syslogd/klogd: done
Starting tcf-agent: [ 17.400862] random: crng init done
[ 17.404276] random: 7 urandom warning(s) missed due to ratelimiting
OK
PetaLinux 2020.1 zcu102_custom_trung /dev/ttyPS0

PetaLinux 2020.1 zcu102_custom_trung /dev/ttyPS0

zcu102_custom_trung login: root
Password:
root@zcu102_custom_trung:~# ls -lh
total 12K
drwxr-xr-x  2 root  root    4.0K Aug 31 02:30 test1_ext4
drwxr-xr-x  2 root  root    4.0K Aug 31 02:30 test2_ext4
drwxr-xr-x  2 root  root    4.0K Aug 31 02:30 test3_ext4
root@zcu102_custom_trung:~# █

```

Hình 5.24. Boot Images thành công khi sử dụng file system trong SD card

## 6. Tổng kết quá trình Booting cho KIT

Quá trình sẽ được chia làm 3 phần bao gồm:

- Bootloader:
  - + Soc ROM stage boot loader
  - + First stage bootloader
  - + Second stage bootloader
- Kernel
- User

### 6.1. Bootloader

- Sau khi được cấp nguồn, hệ thống sẽ bắt đầu thực thi ở reset vector. Lúc này quyền điều khiển sẽ ở vector reset

*Câu hỏi được đặt ra, reset vector là gì?*

- + Reset vector là 1 vị trí mặc định có trong CPU, nơi mà CPU luôn phải tìm đến đầu tiên, nó có nhiệm vụ đi tìm hướng dẫn ban đầu để thực thi sau khi hệ thống reset. Nó có thể là 1 địa chỉ hoặc là 1 con trỏ.
- Sau khi được hướng dẫn, reset vector sẽ trao lại quyền điều khiển cho ROM bootloader.

*Câu hỏi đặt ra, ROM bootloader là gì?*

- + ROM bootloader là một chương trình chỉ đọc, có tác dụng tải khởi động dành riêng cho thiết bị để đảm bảo rằng chip vẫn hoạt động bình thường
- Khi có được quyền điều khiển, ROM bootloader sẽ quyết định đâu là boot device thông qua các jump và pin( USB, SD Card, ...)
- Sau đó ROM bootloader sẽ load The First Stage Bootloader từ boot device vào trong hệ thống và chuyển quyền điều khiển cho FSBL

*Câu hỏi đặt ra, FSBL có tác dụng gì?*

- + FSBL có tác dụng setup bộ nhớ để load image, dọn dẹp lại ổ đĩa và thực hiện tìm kiếm second bootloader ở trong phân vùng FAT của boot device
- FSBL sẽ copy Second bootloader(SSBL) vào trong RAM và chuyển quyền điều khiển cho nó

*Câu hỏi đặt ra, SSBL có tác dụng gì?*

- + SSBL có tác dụng copy dữ liệu khởi tạo đến bộ nhớ của hệ thống, tìm kiếm kernel image và device tree trong thư mục FAT, đọc và load nó vào bộ nhớ ở vị trí 2000:0000
- + Device tree là một kiểu dữ liệu, dùng để mô tả phần cứng để hệ điều hành có thể đọc được mà không cần mã hóa

## 6.2. Kernel

- Sau khi đọc cấu hình cài đặt, SSBL sẽ tìm kiếm linux kernel và device tree binary vào RAM
- Set up kernel boot Argument và chuyển quyền điều khiển cho kernel để nó sử dụng boot args và device tree address để khởi tạo chính nó và thiết bị phần cứng

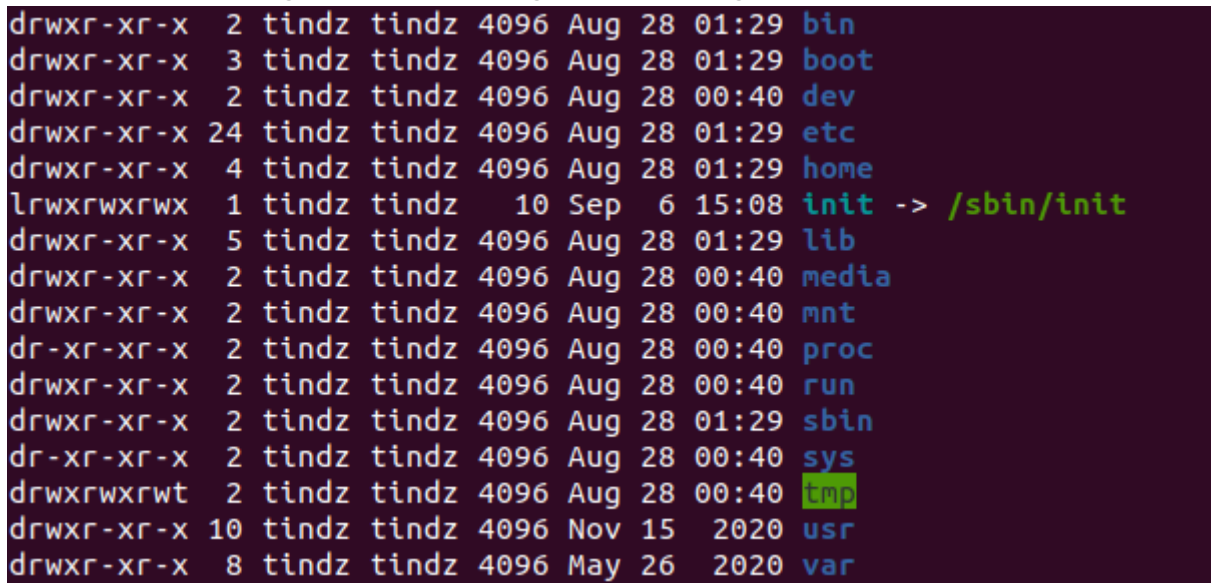
*Câu hỏi đặt ra, kernel boot arguments là gì?*

- + nó là 1 chuỗi ký tự dùng để khởi động kernel, có tác dụng thay đổi các hành vi cụ thể, bật tắt 1 số tính năng nhất định ( ví dụ: đổi password root)
- Sử dụng kernel bootargs, kernel định vị và mounts tới root file system



*Câu hỏi đặt ra, root file system là gì?*

- + hiểu đơn giản thì đây là thư mục gốc của file tree. Nó bao gồm các file và thư mục quan trọng cho OS. Cụ thể là những thư mục thiết bị và chương trình để booting cho hệ thống.



drwxr-xr-x	2	tindz	tindz	4096	Aug	28	01:29	bin
drwxr-xr-x	3	tindz	tindz	4096	Aug	28	01:29	boot
drwxr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	dev
drwxr-xr-x	24	tindz	tindz	4096	Aug	28	01:29	etc
drwxr-xr-x	4	tindz	tindz	4096	Aug	28	01:29	home
lrwxrwxrwx	1	tindz	tindz	10	Sep	6	15:08	init -> /sbin/init
drwxr-xr-x	5	tindz	tindz	4096	Aug	28	01:29	lib
drwxr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	media
drwxr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	mnt
dr-xr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	proc
drwxr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	run
drwxr-xr-x	2	tindz	tindz	4096	Aug	28	01:29	sbin
dr-xr-xr-x	2	tindz	tindz	4096	Aug	28	00:40	sys
drwxrwxrwt	2	tindz	tindz	4096	Aug	28	00:40	tmp
drwxr-xr-x	10	tindz	tindz	4096	Nov	15	2020	usr
drwxr-xr-x	8	tindz	tindz	4096	May	26	2020	var

Hình 4.10. Root File system trong phân vùng rootfs của SD card

### 6.3. User

- kernel sẽ chạy init process để bắt user space
- lúc này init process sẽ sinh sôi ra nhiều user space khác dựa trên những cấu hình file