

۱.

a)

```
for (i=0; i< N; i++)  
    z[i] = a[i] + b[i];  
    w[i] = a[i] - b[i];
```

b)

```
for (i=0; i< N; i++) {  
    x[i] = c[i]*d[i];  
}  
for (i=0; i< N; i++) {  
    y[i] = x[i] * e[i];  
}
```

c)

ابتدا حلقه‌ی بیرونی را distribute می‌کنیم:

```
for (i=0; i< N; i++) {  
    for (j=0; j< M; j++) {  
        c[i][j] = a[i][j] + b[i][j];  
        x[j] = x[j] * c[i][j];  
    }  
}  
for (i=0; i< N; i++) {  
    y[i] = a[i] + x[j];  
}
```

سپس حلقه‌ی داخلی را دو بار distribute می‌کنیم:

```
for (i=0; i< N; i++) {  
    for (j=0; j< M; j++) {  
        c[i][j] = a[i][j] + b[i][j];  
    }  
}  
  
for (i=0; i< N; i++) {  
    for (j=0; j< M; j++) {  
        x[j] = x[j] * c[i][j];  
    }  
}  
  
for (i=0; i< N; i++) {  
    y[i] = a[i] + x[j];  
}
```

۲.

(a) بیشترین دفعات اجرا:

بلوک B1: ۱ بار در ابتدای حلقه اجرا می‌شود.

بلوک B2: ۱۷ بار (از $i = 0$ تا $i = 16$)

بلوک B3: ۱۶ بار (از $i = 0$ تا $i = 15$)

بلوک B4: اگر شرط برقرار باشد، به تعداد B3 اجرا می‌شود (۱۶ بار).

بلوک B5: به تعداد B3 و B4 اجرا می‌شود (۱۶ بار).

(b) کمترین دفعات اجرا:

بلوک B1: ۱ بار در ابتدای حلقه اجرا می‌شود.

بلوک B2: ۱۷ بار (از $i = 0$ تا $i = 16$)

بلوک B3: ۱۶ بار (از $i = 0$ تا $i = 15$)
 بلوک B4: اگر شرط برقرار نباشد، اجرا نمی‌شود (۰ بار).
 بلوک B5: به تعداد B3 و B4 اجرا می‌شود (۱۶ بار).

(c) بیشترین زمان اجرا:

بلوک B1: ۱ بار ≤ 6
 بلوک B2: ۱۷ بار که ۱۶ بار شرط برقرار است $\leq 16 * 2$ و ۱ بار برقرار نیست ≤ 5
 بلوک B3: ۱۶ بار که در تمام آن‌ها شرط برقرار است $\leq 16 * 3$
 بلوک B4: ۱۶ بار $\leq 16 * 7$
 بلوک B5: ۱۶ بار $\leq 16 * 1$

$$6 + 32 + 5 + 48 + 112 + 16 = 219$$

۲۱۹ کلاک طول می‌کشد.

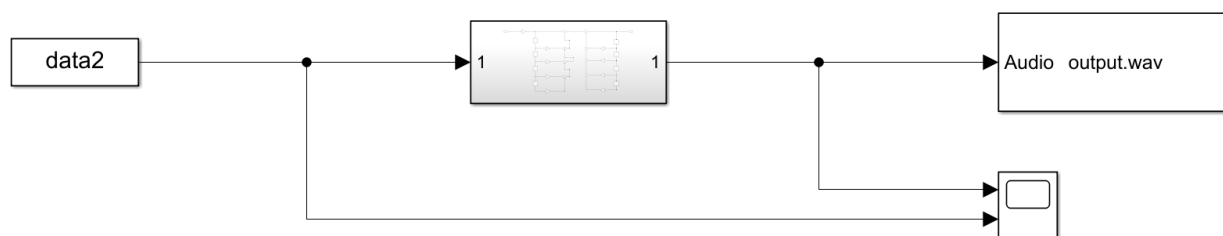
(d) بیشترین زمان اجرا:

بلوک B1: ۱ بار ≤ 6
 بلوک B2: ۱۷ بار که ۱۶ بار شرط برقرار است $\leq 16 * 2$ و ۱ بار برقرار نیست ≤ 5
 بلوک B3: ۱۶ بار که در هیچکدام آن‌ها شرط برقرار نیست $\leq 16 * 6$
 بلوک B4: ۰ بار ≤ 0
 بلوک B5: ۱۶ بار $\leq 16 * 1$

$$6 + 32 + 5 + 96 + 0 + 16 = 155$$

۱۵۵ کلاک طول می‌کشد.

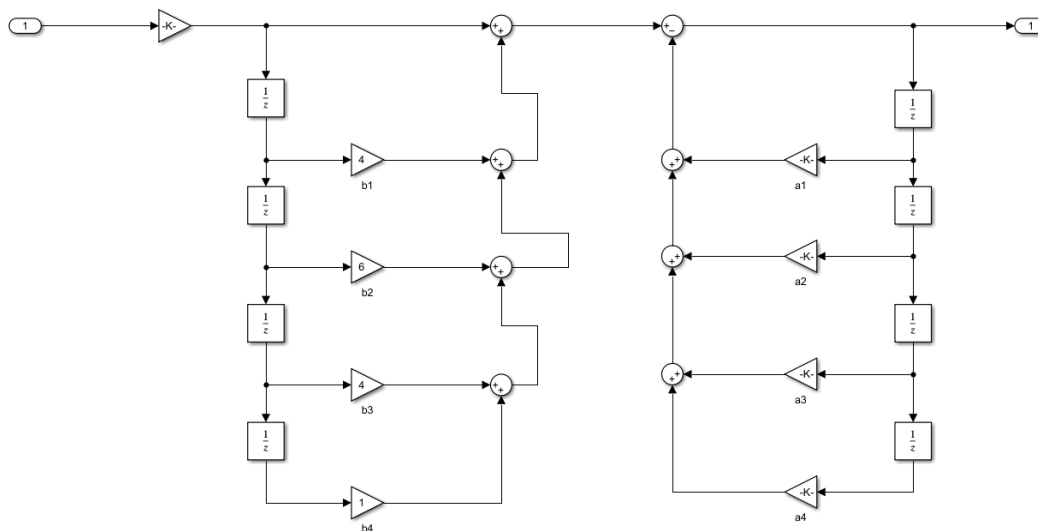
۳. الف و ب) top level این فیلتر به صورت شکل ۱ خواهد بود:



شکل ۱: top level فیلتر توصیف شده

همانطور که در شکل مشاهده می‌کنیم، داده ورودی از طریق بلوک `from workspace` خوانده شده و به فیلتر داده شده و خروجی توسط بلوک `to multimedia file` به یک فایل `.wav` تبدیل شده است. همچنین برای مقایسه مقادیر داده‌های ورودی و خروجی یک `scope` قرار داده شده است.

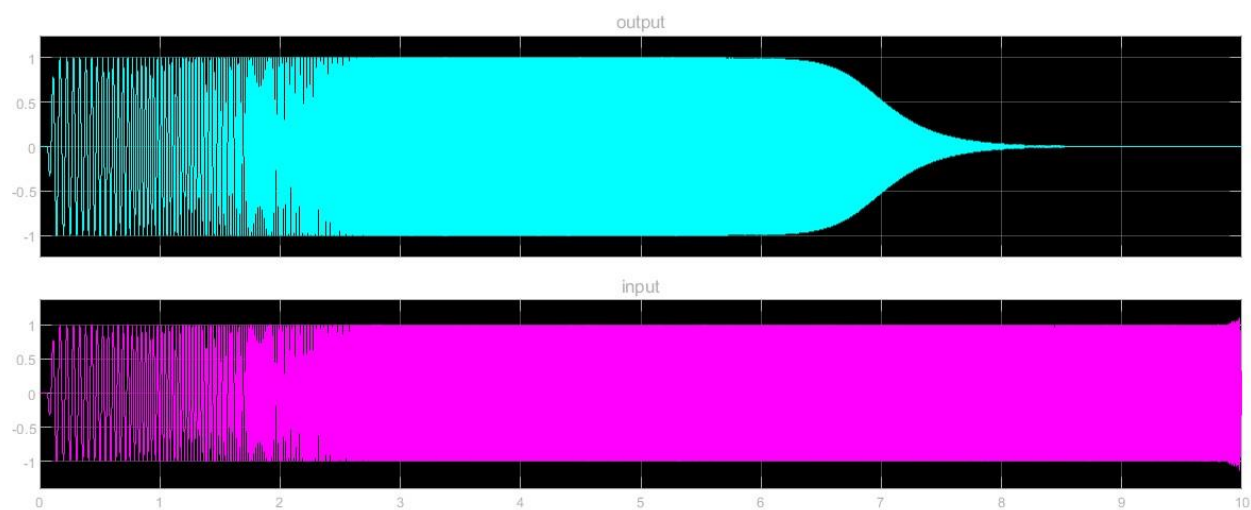
درون `subsystem` به صورت زیر است:



شکل ۲: نمای `subsystem` فیلتر

این فیلتر مشابه توضیحات درون صورت سوال پیاده‌سازی شده است.

حال مدل را ران می‌کنیم. شکل موج ورودی و خروجی به صورت زیر می‌باشد:



شکل ۳: موج ورودی و خروجی مدل

همچنین خروجی این فیلتر با نام `output.wave` درون فایل ارسالی ضمیمه شده است.

ج) با استفاده از ابزار embedded coder کد مدل ساخته شده را برای میکروکنترلر Arduino mega 2560 تولید می‌کنیم. درون تابع `setup()` ابتدا baud rate و مقداردهی اولیه مدل را انجام می‌دهیم. سپس در ۵ بار شبیه‌سازی، مدل را ۱۰۰۰ بار اجرا کرده و زمان اجرای هر سری را حساب می‌کنیم و به تعداد دفعات اجرا (۱۰۰۰) تقسیم می‌کنیم تا میانگین اجرا در هر سری از شبیه‌سازی به دست بیاید.

```
void setup() {  
  
    Serial.begin(9600);  
    q3_p1_initialize();  
    long repeat = 1000;  
    for (size_t i = 0; i < 5; i++)  
    {  
        unsigned long totalTime = 0;  
  
        for (size_t i = 0; i < repeat; i++)  
        {  
            q3_p1_U.Input = random(1, 1000);  
            unsigned startTime = millis();  
            q3_p1_step();  
            totalTime += millis() - startTime;  
        }  
        Serial.print("Simulation ");  
        Serial.print(i+1);  
        Serial.print(" average time: ");  
        Serial.println(totalTime / ((1.0) * repeat));  
    }  
}
```

شکل ۴: کد تابع `setup()`

حال برای اینکه بتوانیم تاثیر سطوح مختلف بهینه‌سازی کامپایلر بر زمان اجرای کد موجود را مشاهده کنیم، از `#pragma once` استفاده می‌کنیم؛ بدین گونه که در ابتدای کد دو دستور `#pragma GCC push_options` و `() #pragma GCC optimize` را قرار داده و در انتهای آن `#pragma GCC pop_options` قرار می‌دهیم. بدین ترتیب با قرار دادن سطح بهینه‌سازی موردنظر در پرانتز، می‌توانیم زمان اجراهای مختلف را بررسی کنیم. در شکل‌های زیر میانگین اجرای برنامه به ازای سطح بهینه‌سازی‌های مختلف مشاهده می‌گردد:

```
Virtual Terminal
Simulation 1 average time: 0.17
Simulation 2 average time: 0.17
Simulation 3 average time: 0.17
Simulation 4 average time: 0.18
Simulation 5 average time: 0.17
```

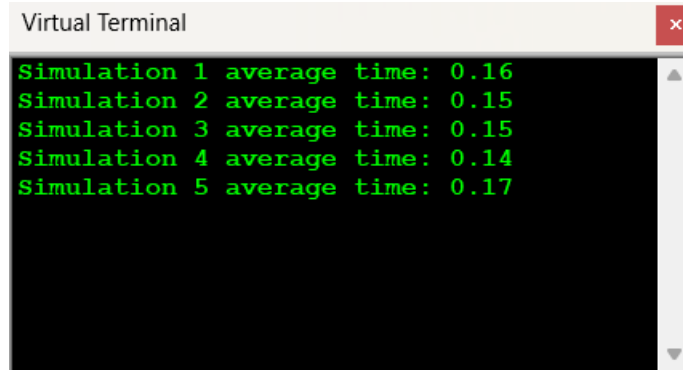
شکل ۵: میانگین زمان اجرا در سطح 00-

```
Virtual Terminal
Simulation 1 average time: 0.15
Simulation 2 average time: 0.15
Simulation 3 average time: 0.15
Simulation 4 average time: 0.14
Simulation 5 average time: 0.16
```

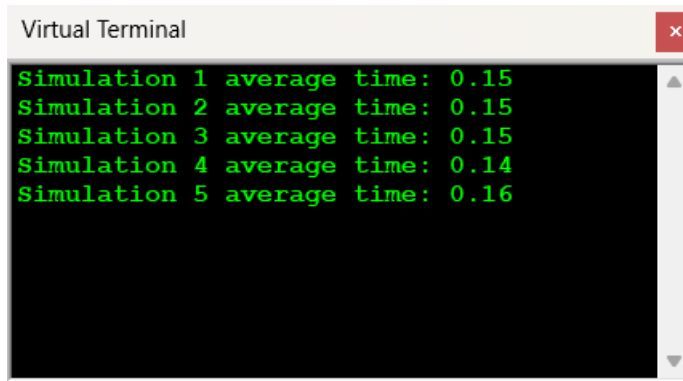
شکل ۶: میانگین زمان اجرا در سطح 01-

```
Virtual Terminal
Simulation 1 average time: 0.16
Simulation 2 average time: 0.15
Simulation 3 average time: 0.15
Simulation 4 average time: 0.14
Simulation 5 average time: 0.17
```

شکل ۷: میانگین زمان اجرا در سطح 02-

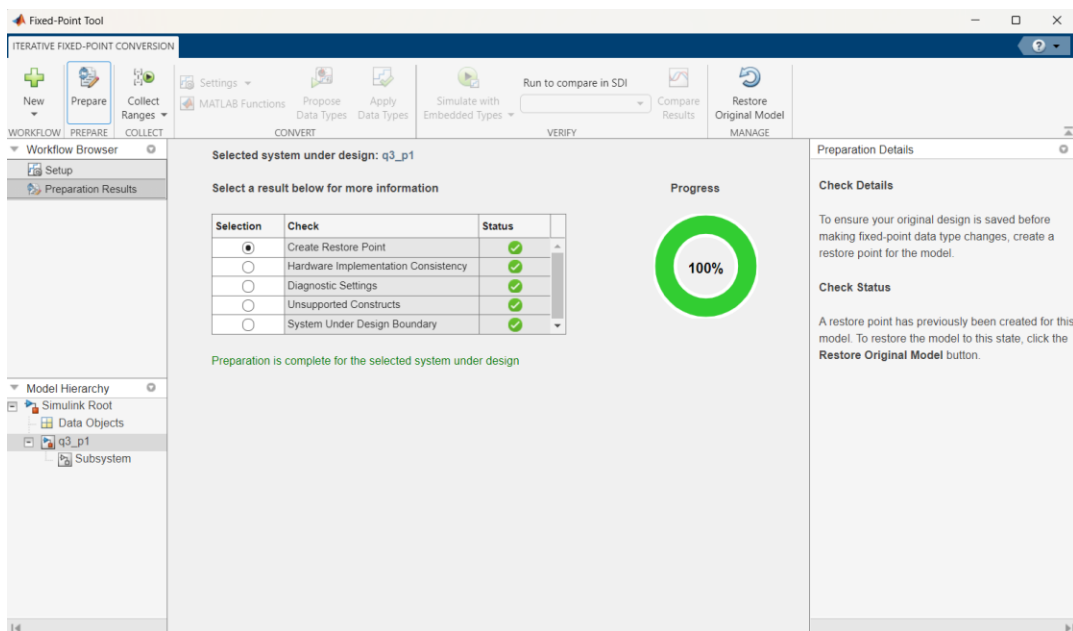


شکل ۸: میانگین زمان اجرا در سطح 03-



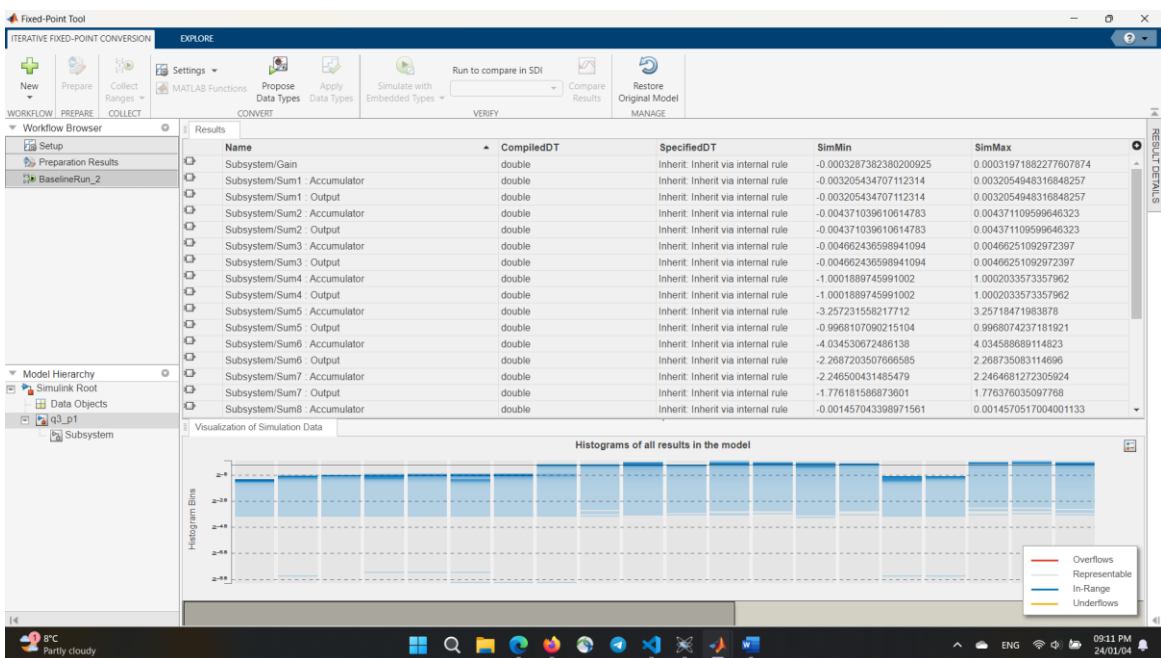
شکل ۹: میانگین زمان اجرا در سطح 05-

د) از بخش apps وارد قسمت Fixed-point Tool می‌شویم و گزینه Iterative Fixed-Point Conversion را انتخاب می‌کنیم. سپس سیستم را prepare می‌کنیم تا مشکلی وجود نداشته باشد.



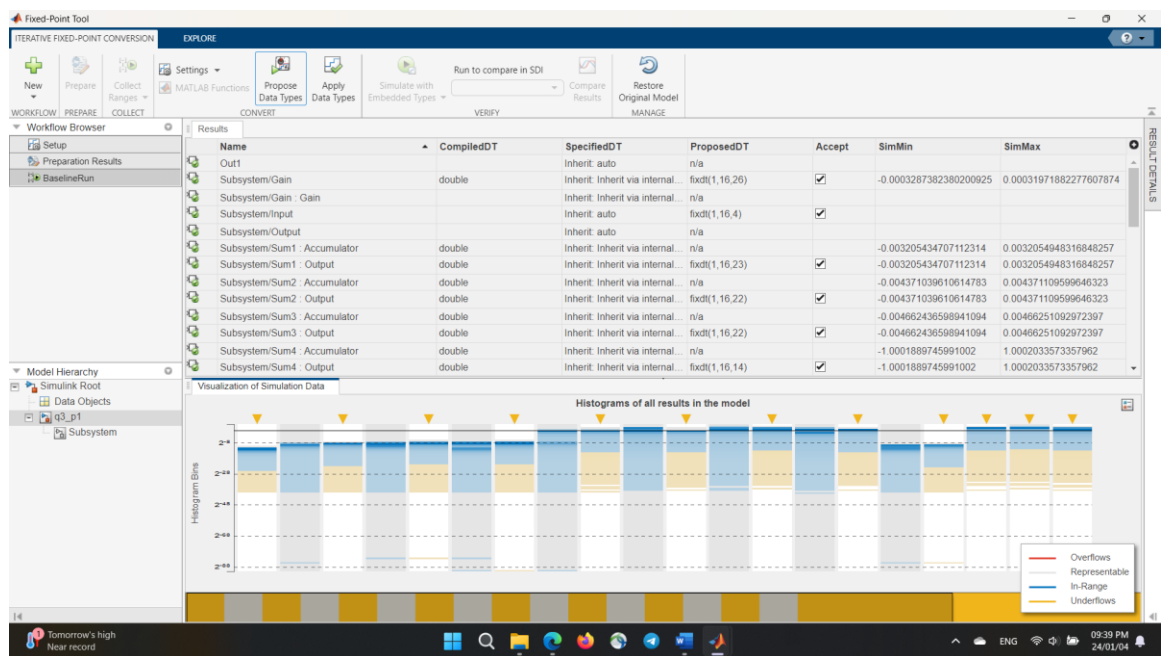
شکل ۱۰: preparation results

حال از بخش collect ranges بر روی گزینه double precision کلیک می‌کنیم و استارت می‌زنیم تا رنج و دقت سیگنال‌ها به دست بیاید. بخشی از این داده‌ها در شکل ۱۱ قابل مشاهده است:



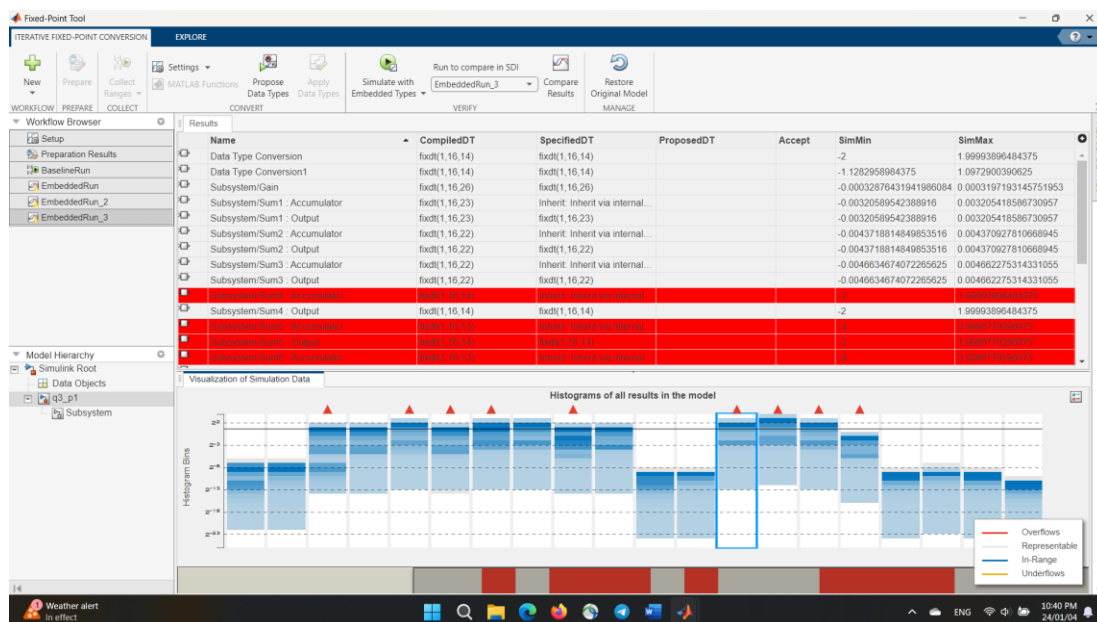
شکل ۱۱: داده خروجی در بخش collect ranges

سپس propose data types اعمال می‌شود. بخشی از این داده‌ها در شکل ۱۲ موجود است:



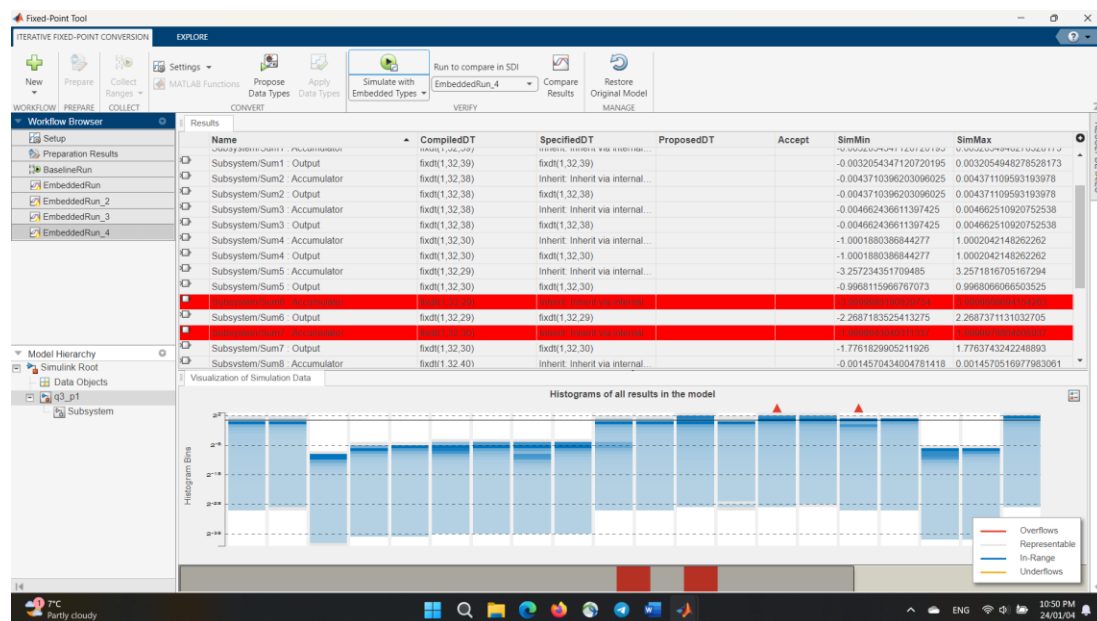
شکل ۱۲: داده خروجی پس از propose data types

سپس دکمه apply data types را میزنیم و سپس با embedded types، شبیه سازی را انجام می دهیم. داده ها به صورت زیر خواهند بود:



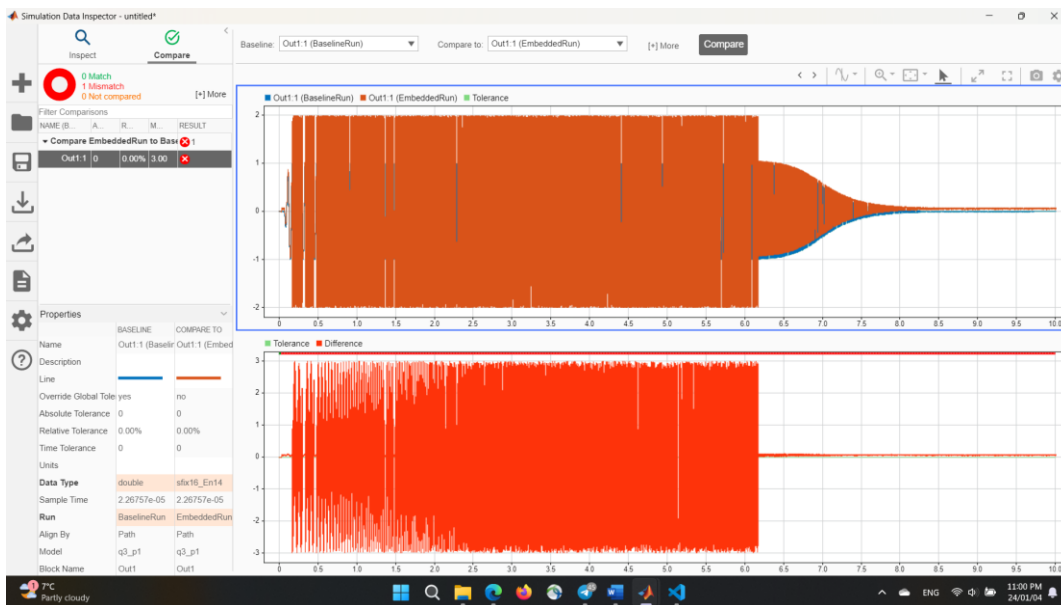
شکل ۱۳: داده نهایی پس از simulate embedded types

همانطور که مشاهده می کنید، در حالتی که داده ها ۱۶ بیتی باشند، در نتایج دچار overflow می شویم. حال اگر داده ها را ۳۲ بیتی کنیم، این مقدار overflow کاسته می شود.



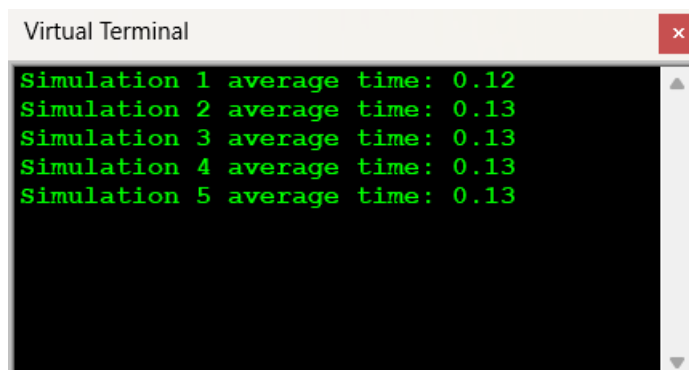
شکل ۱۴: داده ها در حالتی که ۳۲ بیت باشند

همچنین دیتا در حالت embedded run را می توان با حالت baseline run مقایسه کرد که در شکل ۱۵ نشان داده شده است:

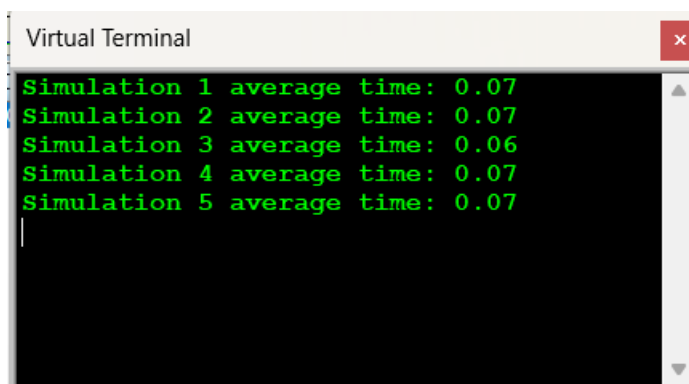


شکل ۱۵: مقایسه دو حالت *baseline run* و *embedded run*

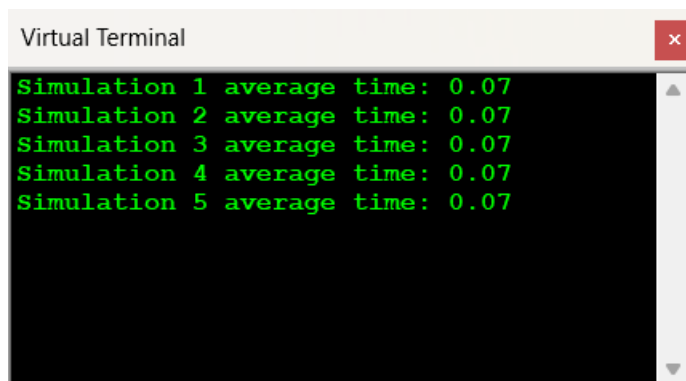
ه) کد را به کمک *embedded coder* تولید کرده و به ازای سطوح مختلف بهینه‌سازی کامپایلر، زمان اجرا را بررسی می‌کنیم.



شکل ۱۶: میانگین زمان اجرا در سطح 00-

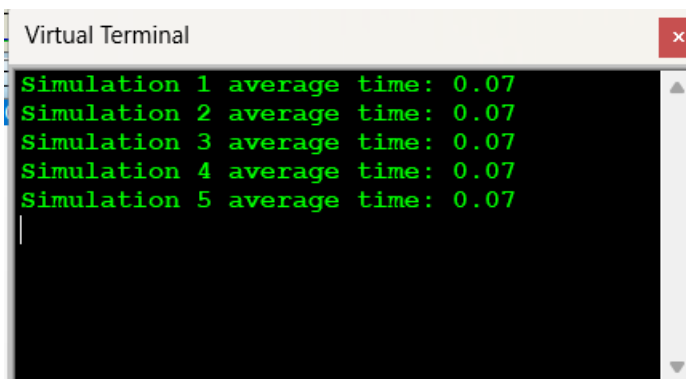


شکل ۱۷: میانگین زمان اجرا در سطح 01-



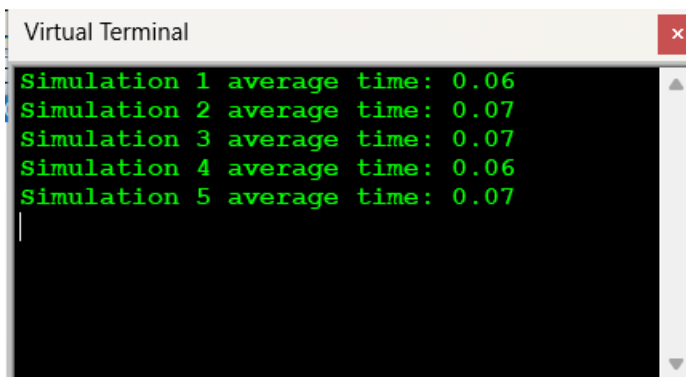
```
Virtual Terminal
Simulation 1 average time: 0.07
Simulation 2 average time: 0.07
Simulation 3 average time: 0.07
Simulation 4 average time: 0.07
Simulation 5 average time: 0.07
```

شکل ۱۸: میانگین زمان اجرا در سطح 02-



```
Virtual Terminal
Simulation 1 average time: 0.07
Simulation 2 average time: 0.07
Simulation 3 average time: 0.07
Simulation 4 average time: 0.07
Simulation 5 average time: 0.07
```

شکل ۱۹: میانگین زمان اجرا در سطح 03-



```
Virtual Terminal
Simulation 1 average time: 0.06
Simulation 2 average time: 0.07
Simulation 3 average time: 0.07
Simulation 4 average time: 0.06
Simulation 5 average time: 0.07
```

شکل ۲۰: میانگین زمان اجرا در سطح 0s-

همانطور که مشاهده می‌شود، متوسط اجرای برنامه با انجام بهینه‌سازی به طور چشمگیری کاهش یافته است.