פרויקט גמר
מערכות משובצות מחשב
פרופ' סיוון טולדו

מגישים:
אושר כהן 038029385
גיא חלאוי 021760624

מאי 2011

אינדקס

1. נושא העבודה

בעבודה זו בחנו את קצב העברת נתונים המתקבלים מדגימות של רכיב ה- ADC ושליחתם לאפליקציית שרת הנמצאת על PC Windows. העברת הנתונים התבצעה דרך פרוטוקול TCP/IP שממומש על הלוח תוך שימוש בחבילת UIP.

דגימות ה-ADC נשמרות במערך ציקלי בצורה סינכרונית (Polling) או אסינכרונית (Interrupt).
שליחת כל חבילת דגימות מתבצעת לאחר קבלת אישור (ACK) על שליחת ההודעה הקודמת.

2. UIP

תשתית התקשורת של הלוח הסתמכה על חבילת UIP (מימוש פרוטוקול TCP/IP על הלוח).
http://www.sics.se/~adam/uip/index.php/Main_Page

2.1 אתחול

שימוש ב- IP סטטי:
192.167.12.1 – כתובת הלוח.
192.167.12.2 – כתובת השרת Windows.

UIP_APPCALL 2.2

פונקציה מרכזית בתשתית UIP, אחראית על ניהול שליחה/קבלה של חבילות לפי מצב הלוח המסומן במשתנה uip_flags (UIP_CONNECTED, UIP_CLOSE, UIP_REXMIT, UIP_ACKDATA וכו').

```
void UIP_APPCALL(void)
{
	// mark if last requrest was handled (got ack)
	static bool lastRequestHandled = 0;

	// connection established
	if(uip_connected())
	{
		// mark connection established
		bConnected = 1;
		printf("connected!!!! ");

		// first send, to recieve an ACK
		uip_send(&dataBuffer[0], ELEM_SIZE);
		return;
	}

	// idle time
	if (uip_poll() && (lastRequestHandled))
	{
		// do not send unless connected
		if (!bConnected)
			return;

		// send data if has something to send
		if (sendInternal())
			lastRequestHandled = 0;

		return;
	}

	// got ack from last request
	if (uip_acked())
	{
		// do not send unless connected
		if (!bConnected)
			return;

		// last request handled
		lastRequestHandled = 1;

		// send next data
		if (sendInternal())
			lastRequestHandled = 0;

		return;
	}

	// pack lost
	if (uip_rexmit())
	{
		if (!bConnected)
			return;

		// resend last request
		printf("Packet Lost! ");
		uip_send(&dataBuffer[startIndex], (lastSent - startIndex) * ELEM_SIZE);

		return;
	}

	// connection closed
	if (uip_closed())
	{
		printf("---socket closed");
		return;
	}
}
```

SendInternal 2.3

קורא לפונקציה UIP_SEND עם הנתונים אותם יש לשלוח. UIP_SEND מעתיקה נתונים אלו למערך אותו תישלח בסוף בחבילה. נתוני הדגימה נשמרים במערך ציקלי ומכאן ששליחתם צריכה להעשות בצורה מיוחדת. הפונקציה תומכת בפסיקת דגימה שתתרחש תוך כדי השליחה.

```
// responsible for sending sampling data
// returns true if data was sent
int sendInternal()
{
	// calculate max data size to send
	int maxDataAmount = uip_mss() / ELEM_SIZE;

	// calculate start index
	startIndex = lastSent % SAMP_BUFFER_SIZE;

	// save end location before changing
	int tmpEnd = endIndex;

	if (startIndex < tmpEnd)
	{
		// send only if has minimum data size to send
		if (tmpEnd - startIndex > maxDataAmount * minDataFraction)
		{
			// has too much data to send
			if (tmpEnd - startIndex > maxDataAmount)
				tmpEnd = startIndex + maxDataAmount;

			// send data
			uip_send(&dataBuffer[startIndex], (tmpEnd - startIndex) * ELEM_SIZE);
			lastSent = tmpEnd;

			return 1;
		}
	}
	else if (startIndex > tmpEnd)
	{
		// has too much data to send
		if (SAMP_BUFFER_SIZE - startIndex > maxDataAmount)
			tmpEnd = startIndex + maxDataAmount;
		else
			// send till end
			tmpEnd = SAMP_BUFFER_SIZE;

		// send data
		uip_send(&dataBuffer[startIndex], (tmpEnd - startIndex) * ELEM_SIZE);
		lastSent = tmpEnd;

		return 1;
	}
	return 0;
}
```

הגדרות נוספות 2.4

- ETHERNET_FRAME_SIZE_MAX – הגדלת גודל ה- frame בשכבת ה- Ethernet.
- UIP_BUFSIZE – הגדלת גודל החבילה הנשלחת

3. ADC

מילוי מערך ציקלי עם דגימות ADC בערוץ מס' 1 (P0.28).
בקשה לדגימה ע"י timer המוקצה לנושא.

בפרויקט בחנו 2 אפשרויות דגימה:

- Interrupt – דגימה אסינכרונית
- Polling – דגימה סינכרונית

3.1 אתחול

```
// init ADC
void ADCInit( DWORD ADC_Clk )
{
    /* all the related pins are set to ADC inputs, AD0.1~4 */
    PINSEL0 |= 0x0F333F00;

    AD0CR = ( 0x01 ) |    // SEL=1,select channel 1 on ADC0
    ( ( 3 ) << 8 ) |      // CLKDIV = Fpclk / 1000000 - 1
    ( 0 << 16 ) |         // BURST = 0, no BURST, software controlled
    ( 1 << 21 ) |         // PDN = 1, normal operation
    ( 0 << 24 ) |         // START = 0 A/D conversion stops
    ( 0 << 27 );          // EDGE = 0 (CAP/MAT singal falling,trigger A/D

    AD0INTEN = 0x11E;       // Enable all interrupts

    // init indexes
    startIndex = 0;
    endIndex = 0;
    lastSent = 0;

    // init ADC interrupt
    vicEnableIRQ(INT_CHANNEL_ADC, 11, ADC0Handler);
}
```

3.2 דגימה באמצעות פסיקה

בקשה אסינכרונית לדגימה:

```
// timer interrrupt
void __attribute__ ((interrupt("IRQ"))) timer_irq(void)
{
      T1IR = BIT0;

      // initiates sampling in channel 1
      if (bConnected)
            ADC0Read(1);

      vicUpdatePriority();
}
```

קריאה של ערכי הדגימה ושמירתם במערך הציקלי:

```
// ADC sampling callback when conversion finished
static void  __attribute__ ((interrupt("IRQ"))) ADC0Handler(void)
{
    DWORD regVal;

    // Read ADC will clear the interrupt
    regVal = AD0STAT;

    //check OVERRUN error first
    if ( regVal & 0x0000FF00 )
    {
        regVal = (regVal & 0x0000FF00) >> 0x08;

        // if overrun, just read ADDR to clear regVal variable has been reused.
        regVal = AD0DR1;

        // stop ADC now
        AD0CR &= 0xF8FFFFFF;
        ADC0IntDone = 1;

        // read data
        dataBuffer[endIndex++] = ADC0Value[1];
        endIndex %= SAMP_BUFFER_SIZE;

        // check for loop
        if ((endIndex == startIndex) && startIndex)
                printf("Buffer Loop! ");

        // Acknowledge Interrupt
        VICVectAddr = 0;
        vicUpdatePriority();
        return;
    }

    if ( regVal & ADC_ADINT )
    {
        if ((regVal & 0xFF) == 0x02)
        {
                // read data
                dataBuffer[endIndex++] = ( AD0DR1 >> 6 ) & 0x3FF;
                endIndex %=  SAMP_BUFFER_SIZE;
        }

        // stop ADC now
        AD0CR &= 0xF8FFFFFF;
        ADC0IntDone = 1;
    }

    // Acknowledge Interrupt
    VICVectAddr = 0;
    vicUpdatePriority();
}
```

3.3 דגימה באמצעות Polling

```
// ADC read using polling
DWORD ADC0ReadPolling( DWORD channelNum )
{
    DWORD regVal, ADC_Data;

    // switch channel,start A/D convert
    AD0CR &= 0xFFFFFF00;
    AD0CR |= (1 << 24) | (1 << channelNum);

    // poll until end of A/D convert
    while ( 1 )
    {
        // read result of A/D conversion
        regVal = *(volatile unsigned long *)(AD0_BASE_ADDR + ADC_OFFSET + ADC_INDEX * channelNum);

        if ( regVal & ADC_DONE )
            break;
    }

    // stop ADC now
    AD0CR &= 0xF8FFFFFF;

    if ( regVal & ADC_OVERRUN )
    {
        // overrun
        dataBuffer[endIndex++] = 0;
        endIndex %= SAMP_BUFFER_SIZE;
        return 0;
    }

    // read data
    ADC_Data = ( regVal >> 6 ) & 0x3FF;
    dataBuffer[endIndex++] = ADC_Data;
    endIndex %= SAMP_BUFFER_SIZE;

    return ADC_Data;
}
```

4. שרת

4.1 קוד

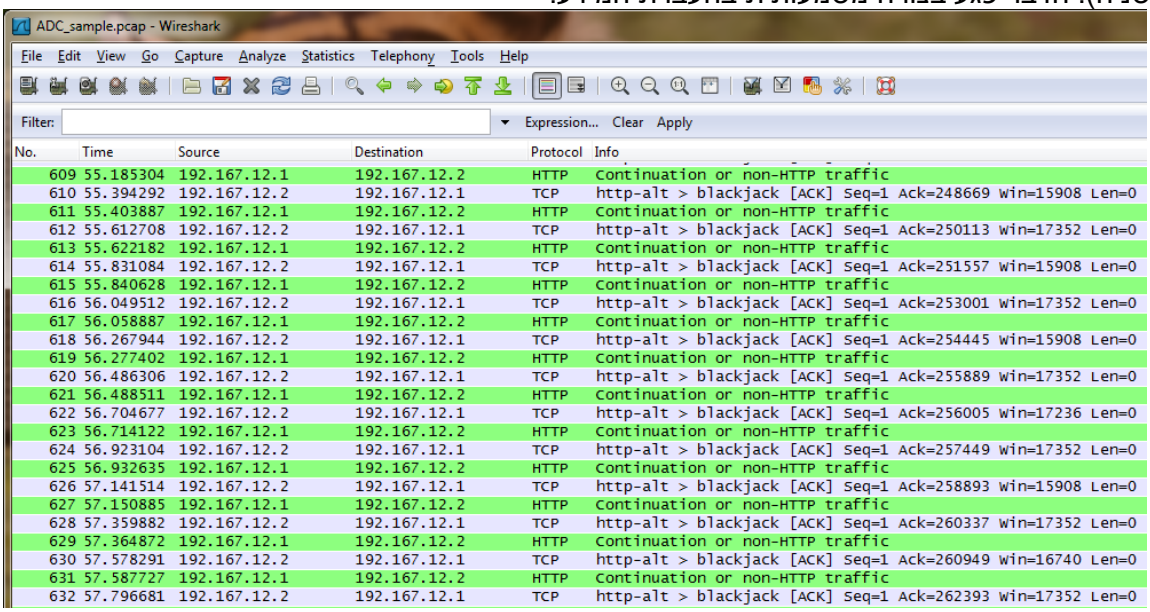
בצד של השרת רצה תוכנית שמקשיבה לפורט 8080, מקבלת את חבילות המידע, שומרת אותן ובסוף הריצה מדפיסה סטטיסטיקות איבוד מידע (ממוצע עבור כל המידע ועבור החצי השני של המידע). כל כפולה של 250 חבילות, התוכנית מדפיסה את קצב העברת המידע הממוצע ואת גודל החבילה הממוצע עבור 250 החבילות האחרונות. עבור כל סט ערכים, הורצה הבדיקה במשך כדקה. לניתוח התוצאות נלקחו הערכים הבאים: קצב העברת המידע וגודל החבילה חושבו מהממוצעים האחרונים בריצה, אחוז איבוד המידע נלקח כאחוז האיבוד בחצי השני של המידע. נעשה שימוש בערך זה מכיוון שבשניות הראשונות של העברת המידע קיימות המון הפרעות והמידע לא עובר בצורה סדירה (הסבר אפשרי לכך הוא כל מיני אתחולים ופעולות חד פעמיות בהקמת הקשר לשרת משבשות את מעבר המידע).
חישוב אחוז איבוד המידע בוצע כאשר לאחר דגימת כל ערך מה- ADC, הכנסנו ערך מספר רץ ל- Buffer, שלחנו אותו ובדקנו לאחר מכן בשרת אלו מספרים קיימים במידע שהתקבל.

```
static void Main(string[] args)
        {
            // listen to port 8080
            TcpListener listener = new TcpListener(8080);
            listener.Start();

            // will save the data
            HashSet<int> wav = new HashSet<int>();  // Debugging
            //List<int> wav = new List<int>();

            while (true)
            {
                // temp buffer for each packet
                byte[] buffer = new byte[10000];

                // accept the socket
                Socket socket = listener.AcceptSocket();
                Console.WriteLine("socket accepted");

                // used later for statistics
                DateTime now = DateTime.Now;
                DateTime start = DateTime.Now;

                List<int> counts = new List<int>();
                List<DateTime> dates = new List<DateTime>();
                int total = 0;
                const int sampleSize = 250;

                while (true)
                {
                    try
                    {
                        int size = socket.Receive(buffer);  // recieve data

                        // save statistics
                        total += size / 4;
                        counts.Add(total);
                        dates.Add(DateTime.Now);

                        // parse each number in data
                        for (int i = 0; i < size; i += 4)
                        {
                            wav.Add(BitConverter.ToInt32(buffer, i));
                        }

                        // print statistics every 'sampleSize' packets
                        int counter1 = counts.Count;
                        if (counter1 % sampleSize != 0)
                            continue;

                        // calc statistics
                        double res = (counts[counter1 - 1] - counts[counter1 - sampleSize]);
                        double res1 = (dates[counter1 - 1] -
                                       dates[counter1 - sampleSize]).TotalMilliseconds / 1000;

                        // print statistics
                        Console.WriteLine(res / res1 + " " + total / counter1);
                    }
```

```
                catch(Exception ex) // occurs when the board 'reset' button is pressed
                {
                    CalculateLoss(wav);    // Debugging

                    // print total size
                    Console.WriteLine("test size = " + wav.Count);
                    wav.Clear();

                    Console.WriteLine("--------------");
                    Console.WriteLine("--------------");
                    break;
                }
            }
        }
        Console.ReadLine();
    }

    private static void CalculateLoss(HashSet<int> wav)
    {
        int max = wav.Max();
        int loss = 0;
        int lostLastHalf = 0;

        for (int i = 0; i < max; i++)
            if (!wav.Contains(i))
            {
                ++loss;
                if (i > max / 2)
                    ++lostLastHalf;
            }

        Console.WriteLine("Loss: {0}, Last Half: {1}", 100 * ((float)loss / (float)max),
                                                       100 * ((float)lostLastHalf / (float)max));
    }
}
```

ACK Delay 4.2

מניתוח התקשורת ב-Wireshark ראינו שיש עיכוב בשליחת ה-Ack ע"י שרת ה-Windows (כחמישית שניה). הדבר פגע בצורה משמעותית בהעברת המידע.



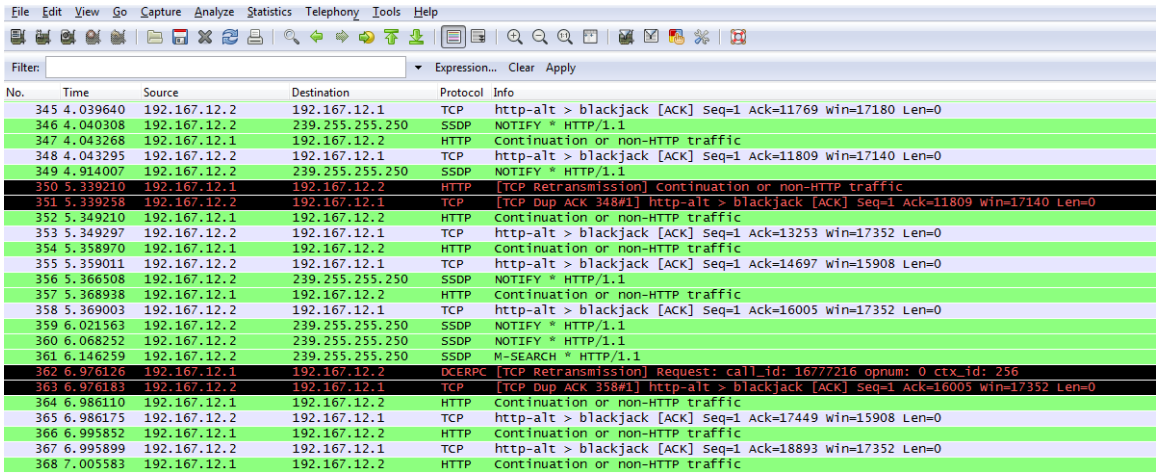
לאחר בדיקה התגלה שברירת מחדל של TCP ב- windows לשליחת ACK היא 200ms על מנת לא להכביד על הרשת. מכיוון שאנחנו שולחים את הבקשה הבאה רק כאשר קיבלנו ACK ברור שקצב השליחה נפגע בצורה משמעותית. על מנת לשנות את ההגדרה ב- Windows צריך לעדכן את ב-registry את הכניסה:

[HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet \Services \Tcpip \Parameters \Interfaces \{Adapter-id}\ TcpAckFrequency]

ערכים אפשריים:
No Delay – 1
Default – 2

Packet Lost 4.3

מניתוח התעבורה ראינו שאנחנו מאבדים מידע רב (Packet Lost) בסמוך להודעות בקרה שנשלחו מ-Windows ללוח. תופעת ה-Packet Lost פגעה משמעותית בביצועים ובאמינות המידע.



על מנת לבטל הודעות בקרה אלו ביצענו את הפעולות הבאות:

1. הורדת Services:
   - SSDP Discovery
   - UPNP Device Host

2. שינוי ב- registry של הערך הבא ל-0:

[HKEY_LOCAL_MACHINE \SOFTWARE \Policies \Microsoft \Tcpip \WindowsNT \DNSClient \EnableMulticast\ ]

3. הורדת IPV6 תחת אפשרויות הכרטיס רשת של הלוח.

4. ביטול NetBIOS תחת אפשרויות מתקדמות של TCP/IP V4 של כרטיס הרשת של הלוח.

MTU 4.4

העלאת גודל החבילה ב- UIP לא שינתה את המצב, גודל החבילות שנשלחו לא היו בגודל המקסימלי האפשרי אלא הרבה פחות. מניתוח התעבורה ב-Wireshark ראינו שה-MTU במהלך יצירת התקשורת עדיין נמוך. בחינה של הקוד העלתה שה-UIP קובע את גודל החבילה לפי המינימום בין הגודל האפשרי שלו (Buffer Size) ובין הגודל שמגיע במהלך יצירת החיבור מהצד השני.
לכן הגדלנו את ה-MTU של השרת בצורה הבאה:

```
C:\TAU\Embedded>netsh interface ipv4 set subinterface "Local Area Connection 2"
mtu=1458 store=persistent
Ok.

C:\TAU\Embedded>netsh interface ipv4 show subinterfaces

   MTU  MediaSenseState   Bytes In  Bytes Out  Interface
------  ---------------  ---------  ---------  -------------
4294967295                1          0    1025608  Loopback Pseudo-Interface 1
  1500                1   12128034    3102137  Wireless Network Connection
  1500                5          0          0  Local Area Connection
  1500                5          0          0  Bluetooth Network Connection
  1458                1    2721800     107073  Local Area Connection 2
```

5. תוצאות

נגדיר פרמטרים בהם השתמשנו בניתוח התוצאות:

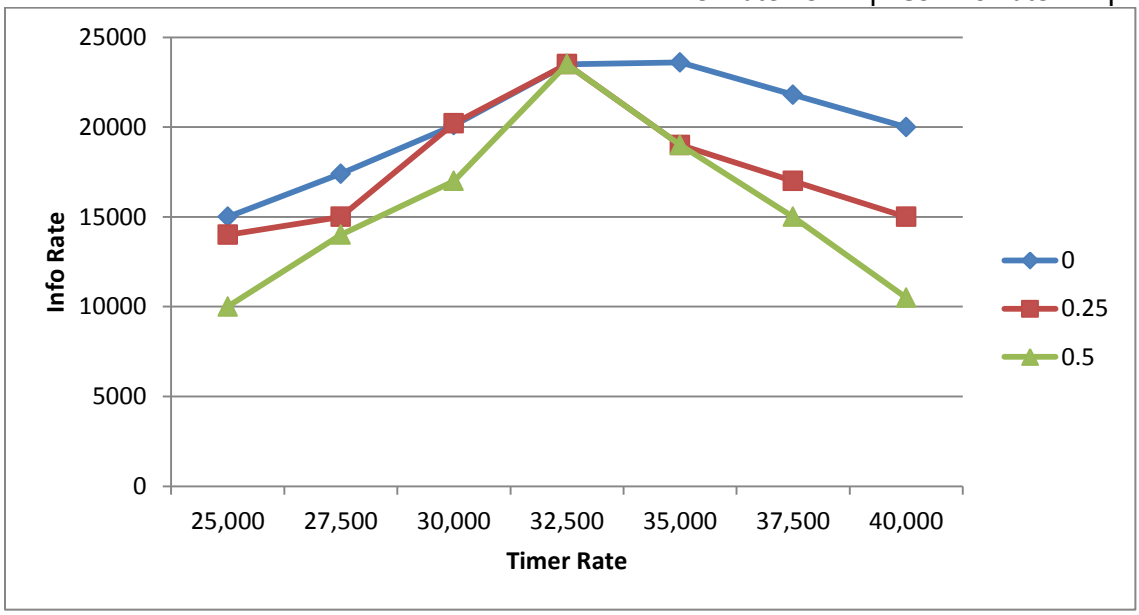
- Frac – מספר בין 0 ל-1 המציין חלק יחסי מגודל החבילה המקסימלי. נאפשר שליחת חבילה רק אם גודלה לכל הפחות מכפלת גודל זה בגודל החבילה המקסימלי.
- Timer Rate – מספר הפעמים בשניה בהן מוקפץ הטיימר המבקש דגימה (polling / interrupt).
- Buffer – גודל המערך בו שומרים את הדגימות לפני שליחתן לשרת.
- Int – ערך בינארי:
  1 מציין דגימה באמצעות פסיקה.
  0 מציין דגימה סינכרונית המחכה עד קבלת הערך (polling).
- Info Rate – קצב המידע המתקבל בצד השרת, נמדד במספרים לשניה.
  גם אם אין איבוד מידע וכל המידע שנדגם נשלח ומתקבל בצד השרת, ערך זה יכול להיות נמוך מערך ה-Timer Rate מכיוון שהלוח מבצע פעולות שונות המשפיעות על קצב הדגימה האמיתי.
- Loss % – אחוז איבוד המידע (מספרים שנדגמו ע"י הלוח ולא התקבלו בצד השרת).
- Packet Size – ממוצע גודל חבילה המתקבלת בצד שרת. גודל מקסימלי לשליחה הוא 361 מספרים (4 / (56 – 1500) = 361, (מספר בתים מקסימלי בחבילה – גודל Header של החבילה / גודל כל מספר בבתים)

5.1 קצב תקשורת כתלות בגודל חבילה מינימלי

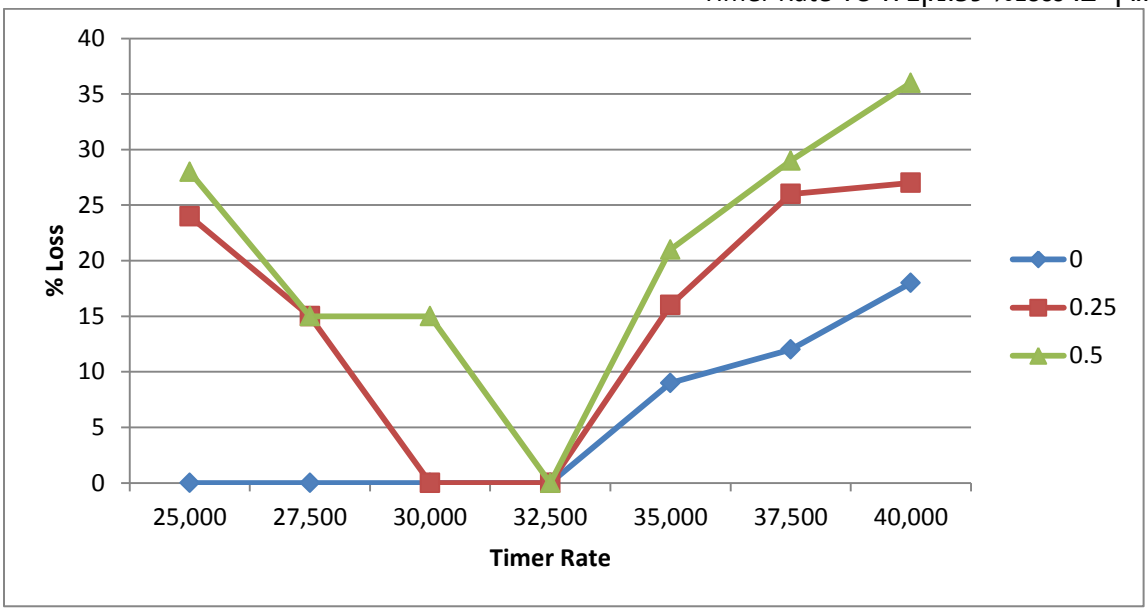
נבחנו שלושה ערכי Frac:

- 0 – חבילה בכל גודל נשלחת לשרת.
- 0.25 – גודל חבילה מינימלית הנשלחת לשרת הינו רבע מגודל החבילה המקסימלית.
- 0.5 – גודל חבילה מינימלית הנשלחת לשרת הינו חצי מגודל החבילה המקסימלית.

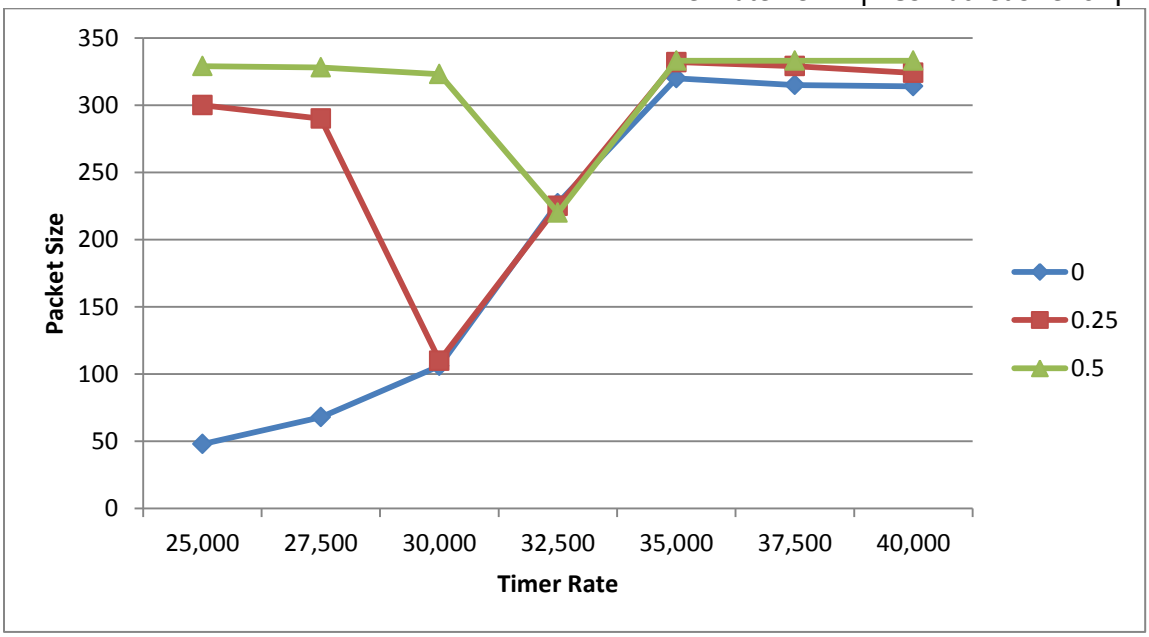
גרף 1: Info Rate כפונקציה של Timer Rate



גרף 2: %Loss כפונקציה של Timer Rate



גרף 3: Packet Size כפונקציה של Timer Rate



- עבור Frac=0 רואים שככל שמגדילים את ה-Timer Rate, כך עולה ה-Info Rate, עד לנקודת מקסימום (32,500) שמעבר אליה ה-Info Rate קטן כתוצאה מאיבוד יותר ויותר מידע.
- ציפינו מראש שקביעת גודל חבילה מינימלי לשליחה ישפר את ה-Info Rate ע"י צמצום מספר החבילות הנשלחות. בשונה מהמצופה אנו רואים כי הגדרת גודל מינימלי של חבילה לשליחה לא משפיעה לטובה על קצב המידע המתקבל בצד שרת.
- ניתן להבחין שבכל המקרים בהם גודל החבילה הממוצעת המתקבלת גדול מ-300 (שליחה של חבילות בגודל מקסימלי + שליחת שארית ה- Buffer כאשר מגיעים לקצה), אחוז איבוד המידע גדל בצורה משמעותית גם אם מדובר בקצבים נמוכים, לדוגמא: Frac = 0.5, Timer Rate = 25K.

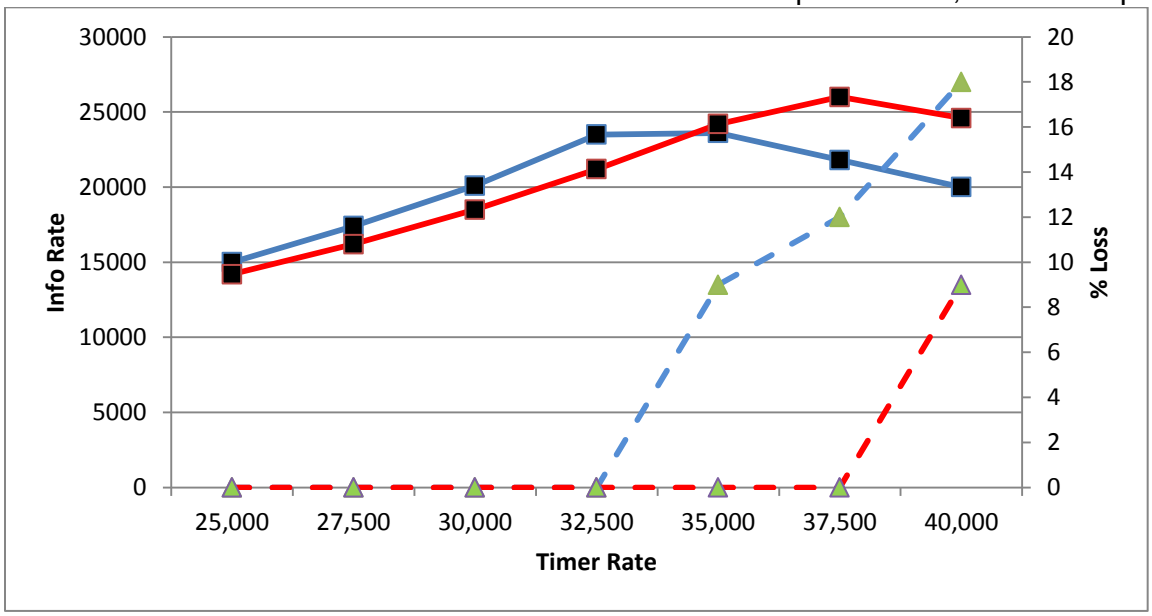
5.2 השוואה בין Interrupt ל- Polling

בגרף הבא מוצגת השוואה בין שתי השיטות לדגימת ערכים מה-ADC:

- כחול – Interrupt
- אדום – Polling

בקווים מלאים מוצגים ערכי ה-info Rate, ובקווים מרוסקים מוצגים ערכי ה-Loss %.
הערכים הבאים התקבלו בהרצות ללא הגבלת גודל חבילה מינימלי.

גרף 1: Info Rate, Loss % כפונקציה של Timer Rate



- מכל הבדיקות שביצענו, ערך ה-Info Rate המיטבי (26,000) התקבל ע"י שימוש ב-Polling עם הפרמטרים הבאים:
  - Timer Rate = 37,500.
  - Frac = 0.
- בדומה לסעיף הקודם, ישנה עליה ב-Info Rate עד לערך Timer Rate מסוים (32,500 עבור Interrupt, 37,500 עבור Polling), וממנו חלה ירידה בגלל איבוד אחוז הולך וגדל של מידע.
- ניתן להבחין שעקומת ה-Info Rate עבור Polling עולה יותר גבוה מזו של ה-Interrupt כתוצאה מאיבוד המידע החל בשימוש ב-Interrupt בערכי Timer Rate נמוכים יותר. הסבר אפשרי לכך הוא שימוש אינטנסיטי בפסיקות, היוצר תקורה גבוהה ומשבש את תהליך העברת הנתונים.

5.3 קצב תקשורת כתלות בגודל ה-BUFFER

בדקנו את השפעת גודל ה-Buffer על קצב המידע המתקבל ואחוז איבוד המידע. לצורך כך בחרנו ארבעה גדלי מערך שונים (1000, 2000, 3000, 4000).

להפתעתנו, גודל ה-Buffer לא השפיע בצורה משמעותית על קצב העברת המידע (גם ב-Polling וגם ב-Interrupt).