ᑭ periodica polytechnica

Electrical Engineering and Computer Science
57/4 (2013) 99–103
doi: 10.3311/PPee.7413

RESEARCH ARTICLE

Communication Time Estimation in High Level Synthesis

György Pilászy / *György* Rácz / *Péter* Arató

Received 2012-09-06, revised 2013-07-29, accepted 2013-07-29

Abstract

The high level synthesis (HLS) tools may result in a multiprocessing structure, where the time demand of the interchip data transfer (briefly the communication) between the processing units (hardware or software) is determined exactly only after the task-allocation. However, a realistic preliminary estimation of the communication time would help to shape the scheduling and the allocation procedures just for attempting to minimize the communication times in the final structure. Compared to the task-execution times of the processing units, especially significant communication times are required by the serial communication interfaces which are frequently used in microcontroller systems. This paper presents an estimation method by analysing four well-known serial communication interfaces (SPI, CAN, I^{2}C, UART).

Keywords

communication time estimation · HLS · CAD · microcontroller · multiprocessing · embedded systems · serial communication interfaces

Acknowledgement

The support of the Hungarian Scientific Fund (OTKA K72611) and New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002) IKT-P5-T3 are gratefully acknowledged. This work is also belonging to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project.

György Pilászy
Department of Control Engineering and Information Technology, BME
e-mail: gpilaszy@iit.bme.hu

György Rácz
Department of Control Engineering and Information Technology, BME
e-mail: gyuriracz@iit.bme.hu

Péter Arató
Department of Control Engineering and Information Technology, BME, Magyar tudósok krt 2., H-1117 Budapest
e-mail: arato@iit.bme.hu

1 Introduction

In high-level design, the task-specification can be transformed into some kind of data flow graphs. Various HLS (High Level Synthesis) algorithms and tools are available for optimizing the schedule and allocation of the graph. [3, 7, 9, 11–15]. The HLS framework presented in [7] for synthesizing a specific (task-dependent) multiprocessing structure demonstrates that how important is a realistic preliminary communication time estimation already in the decomposition phase. The HLS tools are rarely dealing with the communication between the nodes of the data flow graph; it is generally considered with zero-time execution. This solution is appropriate within an intra-FPGA (Field Programmable Gate Array) communication, but in case of more than one IP (Intellectual Property Unit or Intelligent Processor) it is not always applicable. In IPs containing microcontrollers or microprocessors, generally integrated communication peripherals are applied. Compared to the task-execution times of the processing units, especially significant communication times are required by the serial communication interfaces which are frequently used in microcontroller systems.

If the task-specification allows, each communication channel (line) can be represented by an extra node with an estimated or determined execution time representing the communication time.

A realistic preliminary estimation of the communication time would help to influence the scheduling and the allocation procedures in order to reduce the communication complexity between the processing units in the final structure.

Further on, we present an estimation method by analyzing four well-known serial communication interfaces (SPI, CAN, I^{2}C, UART).

2 Calculating the communication time

Various types of serial channels are often used, because of the small number of pins and other resource constraints. In the following sections, we examine the frame structures of four frequently integrated serial communication interfaces.

2.1 Analysis of the simple interfaces (without protocol)

In the following, we calculate the communication time of the well known SPI and UART interfaces. These interfaces transmit data in typically 8 bit units and only specify timing requirements.

Figure 1 and 2 show the timing diagrams of the data transfer. Based on these diagrams the communication time (T_k) can be easily calculated.

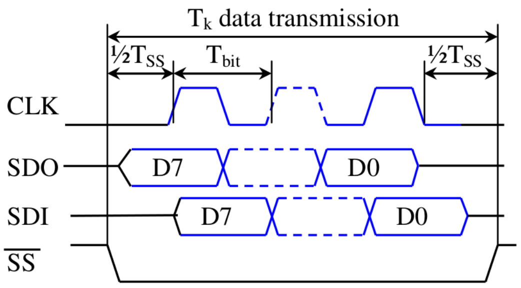

Fig. 1. SPI data transmission

$$T_k = b \cdot T_{bit} + T_{SS} \tag{1}$$

$$T_{SS} =\approx T_{bit} \tag{2}$$

$$T_k = (b + 1) \cdot T_{bit} \tag{3}$$

$$T_k = (N \cdot 8 + 1) \cdot T_{bit} \tag{4}$$

where T_k : communication time, T_{bit} : clock period, T_{SS} : sum of the selection/deselection time, b : the number of bits to be transmitted, N : number of bytes.

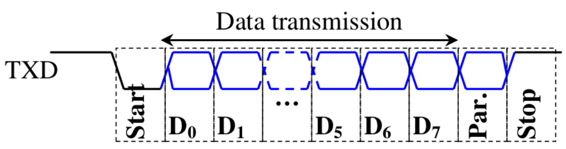

Fig. 2. UART data transmission

$$T_k = (b + 1 + p + s) \cdot T_{bit} \tag{5}$$

$$T_k = N \cdot (1 + 8 + p + s) \cdot T_{bit} \tag{6}$$

where p : parity or control bit (0 or 1), s : stop bits (0, 1, 1.5, 2).

These interfaces can operate in wide ranges of clock frequencies. The typical upper limit is 20MHz in case of SPI and 10MHz in case of UART. The values mostly depend on the transmission medium.

2.2 Analysis of an interface with simple protocol

In this case the specification of the interface contains some additional overhead bits and signals. A widespread example of such an interface is the Inter-IC bus [8]. Figure 3 shows the timing diagram of an I^{2}C communication with 7 bit addressing. Based on [8] and figure 3 the communication time of the I^{2}C bus can be formulated.

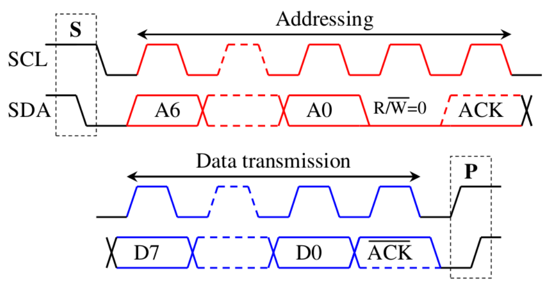

Fig. 3. I^{2}C data transmission with 7 bit addressing

Assuming that we fix the time of the START and STOP bits as one bit period each, we can state concerning the number of the bits to be transmitted shown in Table 1.

Tab. 1. Number of the bits to be transmitted

	7bit address	10bit address
framing	2 bit	2 bit
address	9 bit	9+9 bit
databits	$N \cdot 9$	$N \cdot 9$
Total transmitted bits	$11 + N \cdot 9$	$20 + N \cdot 9$

Typical I^{2}C bus speeds [4] are shown in Table 2.

Tab. 2. Typical I^{2}C speeds and bit times. *Note: The "high speed" modes require special handling [4].

Speed mode	Max bitrate [kbit/s]	Bit time (T_{bit}) μs
Normal	100	10
Fast	400	2.5
Fast +	1000	1
High speed	3400*	0.29
Ultra high speed	5000*	0.2

Table 8 shows the estimated communication times of the I^{2}C bus. Assuming 100 kHz clock frequency and normal 1-8 data bytes, the message transmission periods are shown in the Table 3.

Tab. 3. Message transmission periods in μ s

Databytes (N)	7 bit address	10 bit address
1	200	290
2	290	380
3	380	470
4	470	560
5	560	650
6	650	740
7	740	830
8	830	920

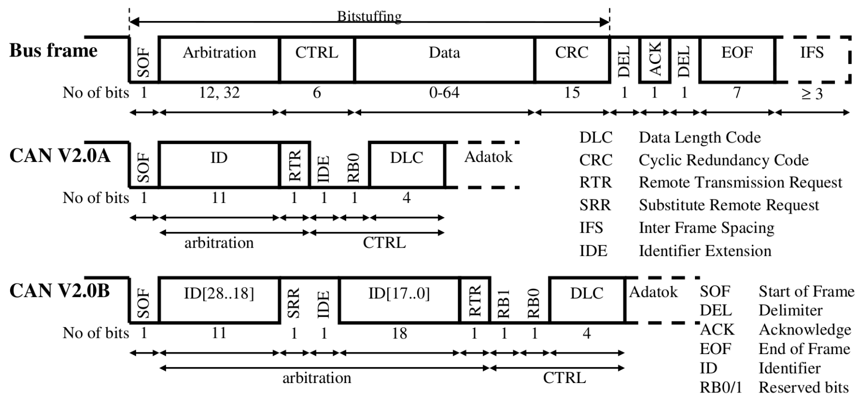

Fig. 4. Structure of CAN dataframes [1, 2]

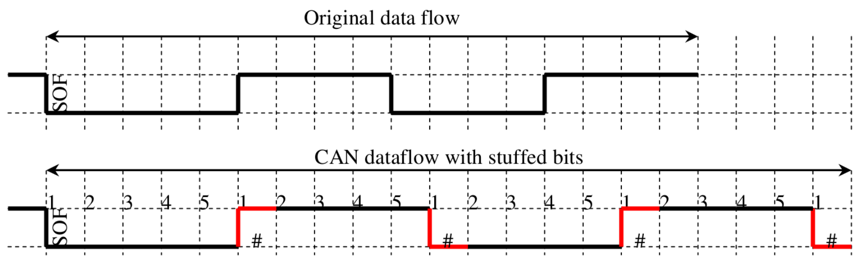

Fig. 5. Bitsuffing

2.3 Analysis of an interface with multi layer protocol

The CAN (Control Area Network) is a multi layer serial communication protocol [1] which supports the real-time distributed control, at up to 1 Mbit/s speed. There are two types of CAN protocols: CAN V2.0A (standard format) which uses 11 bits, and the CAN V2.0B (extended format) which uses 11+18 = 29-bits as identifier in the transmission of messages. The devices, which can use the extended format, are able to communicate in the standard mode too. Thus, the different devices can work together, but only in the standard mode.

The structure of CAN messages are carefully detailed in [1] and [2].

Figure 4 shows the structure of the CAN data frame. At the three interfaces described above, the length of the message did not depend on the content. Because of the so called bitstuffing applied in the CAN system we can give only a worst case estimation for the communication time. We must estimate also the maximum bit number of the longest CAN message.

The length of the CAN message can vary depending on the transmitted data and the stuffed bits. The method of the bitstuffing is as follows: if the transmitter detects five consecutive bits of identical value in the bit stream. Figure 4 shows the structure of the CAN data frame. At the three interfaces described above, the length of the message did not depend on the content. Because of the so called bitstuffing applied in the CAN system we can give only a worst case to be transmitted it automatically inserts a bit of opposite value in the actual bit stream before the transmission [2]. The frame segments, start of frame, arbitration field, control field, data field and CRC sequence are coded by the method of bit stuffing [2].

Tab. 4. Maximal bit number of CAN messages

	CAN 2.0A	CAN 2.0B
Fix fields	34 bits	54 bits
N [byte]	Total length [bit]	
1	53+13	78+13
2	63+13	88+13
3	73+13	98+13
4	83+13	108+13
5	93+13	118+13
6	103+13	128+13
7	113+13	138+13
8	123+13	148+13

To estimate the maximum number of the inserted bits, ignore the fixed control bit values (eg.: IDE, RB0, RB1, etc). The frame starts with a zero SOF bit, so we start also the sample sequence identifier with 0 bits. After five 0 bits (including the SOF), a 1-bit value will be inserted. Afterwards, the test series continues with 4 additional 1-bits, then again a 0 bit, and so on. Figure 5 shows the first 16 bits of the test sequence.

Of the above test series can be seen that for the transmission of 16 bits 4 stuffed bits were needed. The "#" character marked the inserted bits.

The maximum number of stuffed bits (s) can be estimated by dividing the bit number of the message bits by 4, and then the

Tab. 5. Maximal CAN transmission time

Message size	CAN 2.0A	CAN 2.0B
byte	μs	μs
1	66	91
2	76	101
3	86	111
4	96	121
5	106	131
6	116	141
7	126	151
8	136	161

result is rounded up.

$$s = \left\lceil \frac{\text{No of databits}}{4} \right\rceil = \left\lceil \frac{FM + AM}{4} \right\rceil \tag{7}$$

where FM : bit number of fix fields, AM : bit number of variable data field, $AM = 8 \cdot N$, where N is the number of data bytes.

According to the above considerations, the total number of bits in the frame (db) is as follows:

$$db = FM + AM + s + E \tag{8}$$

where E : number of non stuffed bits after the CRC at the end of the frame, $E \geq 13$.

Taking into account the bit number of the fields (in the fix header and the variable lengths of data and the empty space (13 bits) at the end of frame and the stuffed bits), the following maximum number of bits can be calculated shown on Table 4.

2.4 Transmission time estimation of the CAN interface

The maximum transmission time can be estimated, if we know the bit-time of the CAN interface. For this kind of estimation, the previously calculated message length is multiplied by the transmission time of one bit. Reordering the equation (8):

$$db = FM + \left\lceil \frac{FM}{4} \right\rceil + E + AM + \left\lceil \frac{AM}{4} \right\rceil \tag{9}$$

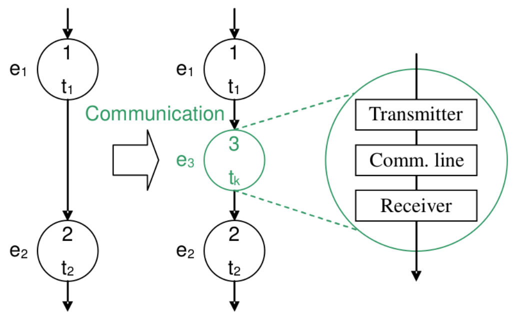

Fig. 6. DFG representation

As AM field size can be 0. . .8 bytes, the inserted bits can be maximum 2 of each byte. When N is the number of data bytes, the total bit number of the data field is:

$$AM + \left\lceil \frac{AM}{4} \right\rceil = N \cdot (8 + 2) = N \cdot 10 \tag{10}$$

Tab. 6. The input parameters

Processor	Operation	t_i	pieces
P1	FFT	99	4
P2	SC	29	4
Pk	CAN	18	4
P4	HT	111	1

For example: at 1Mbit/s data transfer rate from Table 4 is shown in Table 5.

By using the closed forms, the results presented above are summed up in Table 8.

3 Using the results in HLS design systems

The HLS tools usually represent the task to be solved by forming a data flow graph (DFG). The Figure 6 shows a simple example, how to represent the communication channel (e_3) between the e_1 and e_2 elementary operations. In Figure 6, t_1 and t_2 represent the duration of the operation e_1 and e_2.

The execution time of the e_3 communication operation is t_k. However, the communication time (T_k) presented in the previous chapters depends on the bit rate, so we have to transform it into the timing system used by the particular HLS tool. The communication time is characterized by the T_{bit} and T_k. In the HLS tools the time is generally modelled by the number of the clock periods. The two time domains should be scaled as follows:

$$t_k = \frac{T_k}{T} \tag{11}$$

where T denotes the length of the clock period and t_k is T_k expressed by the number of clock periods.

As an example for the practical usage of this method, we have chosen a sound source localisation structure from the reference [6]. Let a CAN communication network be assumed between the microcontrollers. In order to optimize the graph structure, we used the HLS tool PIPE [3]. The Elementary Operation Graph (EOG) of the task is shown in Figure 7 on the left. By applying the tool PIPE for scheduling and allocation, the allocated DFG is illustrated in Figure 7 on the right.

The execution times of the operations are assumed as shown in Table 6.

In the example, it has been assumed that a restart time (initialisation period) $R = 260$ clock period ensures the desired pipeline throughput. To fulfil this requirement, the necessary numbers of operations are summarized in Table 7.

Tab. 7. Resources after the allocation

Processor	Operation	t_i	pieces
P1	FFT	99	2
P2	SC	29	2
Pk	CAN	18	1
P4	HT	111	1

Tab. 8. Communication time estimation

Interface	T_k
SPI	$(N \cdot 8 + 1) \cdot T_{bit}$
I^2C-7 bits address	$(N \cdot 9 + 11) \cdot T_{bit}$
I^2C-10 bits address	$(N \cdot 9 + 20) \cdot T_{bit}$
UART (1 stop, 0 parity)	$N \cdot 10 \cdot T_{bit}$
UART (1 stop, 1 parity)	$N \cdot 11 \cdot T_{bit}$
CAN2.0A	$(N \cdot 10 + 56) \cdot T_{bit}$
CAN2.0B	$(N \cdot 10 + 81) \cdot T_{bit}$

The results show that the HLS tool PIPE provided only two FFT blocks and only one CAN interface at the specified restart time.

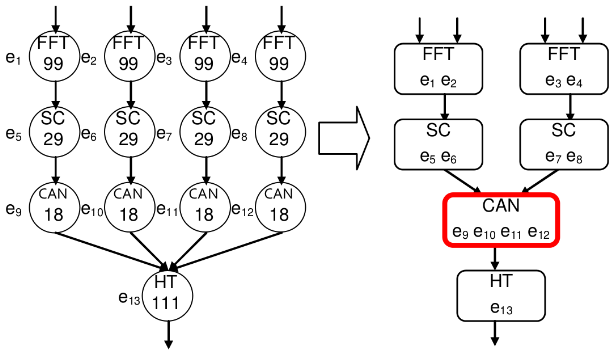

Fig. 7. The EOG (left) and the allocated DFG (right)

4 Results

The method presented in this paper can be applied to estimate the communication time in four frequently serial communication interfaces. Since such interfaces are byte-organized in most cases, so it is advisable to indicate the data to be forwarded (N) in bytes.

The results are summarized in Table 8. N is the number of bytes ($N = 1 \ldots 8$), T_{bit} is the bit time. The $N \leq 8$ limit is needed, because the maximum size of CAN messages can be 8 bytes. Figure 8 shows the message transmission time of various communication interfaces at $T_{bit} = 1\,\mu$ s bit time. The presented method can be applied in other types of interfaces as well.

References

1 *CAN specification 2.0A*, `http://www.can-cia.org/index.php?id=441`.

2 *CAN specification 2.0B*, `http://www.can-cia.org/index.php?id=441`.

3 **Arató P, Visegrády T, Jankovits I**, *High Level Synthesis of Pipelined Datapaths*, John Wiley & Sons; New York, 2001, ISBN 0 471495582 4.

4 *I2C-bus specification and user manual Rev. 4*, 2013, February, `http://www.nxp.com/documents/user_manual/UM10204.pdf`.

5 **Pilászy G, Móczár G**, *Remote control of modular microcontroller systems*, In: Measurement and Automation, Microcad 2001; Miskolc, Hungary, 2001.03.01-2001.03.02.

6 **Goraczko M, Liu J, Lymberopoulos D**, *Energy-Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems*. DAC 2008 (June 8–13, 2008, Anaheim, California, USA).

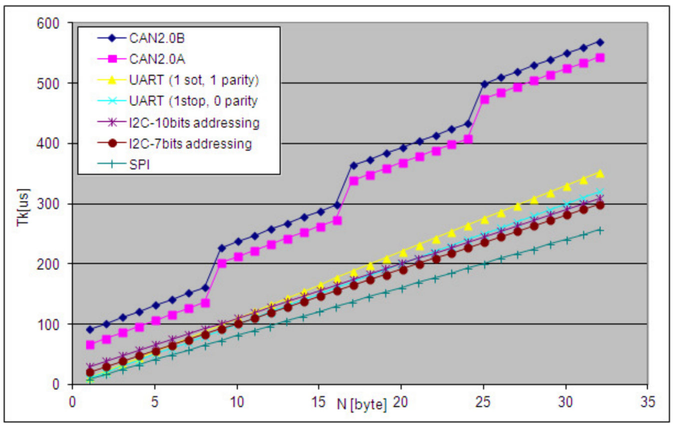

Fig. 8. Comparison of the communication times at $T_{bit} = 1\mu$s

7 **Arató P, Drexler D, Kocza G, Suba G**, *Synthesis of a Task-dependent Pipelined Multiprocessing Structure*, ACM Transactions on Design Automation of Electronic Systems.

8 *The I2C Bus specification version 2.1*, Jan. 2000, `http://www.nxp.com/documents/other/39340011.pdf`.

9 **Coussy P, Gajski DD**, *An Introduction to High-Level Synthesis*, IEEE Design & Test of Computers, (2009).

10 *8251A Programmable communication interface*, Intel Corporation, 1993. Document number: 205222-003.

11 **Suba G**, *Hierarchical pipelining of nested loops in high-level synthesis*. submitted to Periodica Polytechnica Electrical Engineering and Computer Science.

12 **Hou J, Wolf W**, *Process Partitioning for Distributed Embedded Systems*, In: Proceedings of the 4th International Workshop on Hardware/Software Co-Design, IEEE, 1996, p. 70, DOI 10.1109/HCS.1996.492228.

13 **Reinhard W, Grund D, Reineke J, Schlickling M, Pister M, Ferdinand C**, *Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems*, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, **28**(7), (2009).

14 **Séméria L, Sato K, De Micheli G**, *Synthesis of hardware models in c with pointers and complex data structures*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems - System Level Design, (2001).

15 **Schliecker S, Rox J, Negrean M, Richter K, Jersak M, Ernst R**, *System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures*, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, **28**(7), (2009).