Investigation of the speed-up of a dual microcontroller parallel processing system in the execution of a mathematical operation

Peter Harrington
School of Computing, Engineering and Information Sciences, Northumbria University
Newcastle upon Tyne, U.K.
peter.harrington@northumbria.ac.uk

Wai Pang Ng
School of Computing, Engineering and Information Sciences, Northumbria University
Newcastle upon Tyne, U.K.
wai-pang.ng@northumbria.ac.uk

Abstract— **An investigation of the performance of a two microcontroller parallel processing system is presented. A two-microcontroller parallel processing is developed using low end microcontrollers (PIC 16F877). An 8x8 bit multiply operation and a 16x16 bit multiply operation are executed on a single microcontroller and on the proposed dual microcontroller parallel processing system in order to assess the performance of the proposed system. Results presented show poor performance for the 8x8 bit multiply with an average speed up factor of 0.82 This is due to the time required to transfer data around the dual microcontroller system being significant in comparison to the time required to complete the multiply operation, thus nullifying the potential advantage that might be expected of a dual microcontroller system. The 16x16 multiplier exhibited good performance, with results showing a maximum average speed up factor of 1.7 and an average speed up factor of 1.5. The 16x16 multiplication requires longer time to compute and the data transfer time between microcontrollers whilst still having an impact on the overall computation time is significantly less than for the 8x8 multiplier A formula has been developed to provide an estimate of the possible speed up within a system in relation to the process execution time and the time required to communicate data around the proposed system. The proposed system was developed and tested using the Proteus simulation software.**

I. INTRODUCTION

Speed-up or improved performance, in terms of program execution time, of a computer system can no longer be achieved with a single processor [1]. Adopting systems where sections of code can be executed by several or more processors in parallel is the only way that performance can be significantly improved. Since the development of the 8-bit microprocessors of the late 1970s, manufacturers have made improvements in performance by incorporating elements of parallelism into their processors. Expansion of data bus width improved program fetch cycle time from memory improving execution times by up to a factor of four. Advances in manufacturing techniques allowing a greater number of more complex circuitry and interconnections to be placed on the silicon itself led to the development of the Harvard architecture [2] from the original Von Neumann architecture [3]. The Harvard approach involved separate address and data busses to program and data memory which allows the program memory bus width to accommodate the width of a single processor instruction thus allowing the instruction to be extracted from memory in one fetch cycle. Harvard architecture enables a further enhancement to performance known as pipelining, which allows fetching of the next instruction and execution of the current instruction to occur simultaneously. Thus an instruction can be executed every instruction cycle as is the case with the low end Microchip Technology microcontrollers [4].

For future enhancements the only way to significantly improve performance is to move away from internal microprocessor architecture development, and move towards the development of parallel microprocessor structures incorporating multiple microprocessors [1]. Sakamoto and Hase [5] show that parallel execution of arithmetic and accumulator operations can increase processing speeds by up to 35%. Maslennikov *et al* [6] implement a processor array structure with a common bus architecture to RAM producing a six times execution speed improvement compared to the single processor configuration. Schubert and Becker [7] produce a multi-microcontroller system which speeds up SAT algorithms and Bin [8] develops a dual microcontroller based GPRS data transmission control system design which processes events more quickly than a single microcontroller solution.

Within a computer program there will be serial elements and there will be parallel elements. Amdahl [9] offers a prediction for the speedup factor of an algorithm depending upon the serial and parallel content and the number of processors in the system. This speedup factor $S(n)$ is defined as [9]

$$S(n) = \frac{t_p(1)}{t_p(n)} \qquad (1)$$

where $t_p(1)$ is the process time on a single processor and $t_p(n)$ is the process time using n parallel processors. It is the speedup factor that it the whole purpose behind the development of parallel processing systems. Under ideal circumstances and for a fully parallelizable algorithm

$$S(n) = n \qquad (2)$$

Unfortunately this is never the case. A parallel processor system will consist of a network of processors and memory, all of which must communicate with each other.

This investigation involves the development of a parallel processing system incorporating two microcontrollers to ascertain whether speed up is possible within this type of architecture, and to attempt to quantify the ratio in terms of the time required to complete a process and the time required to communicate data around the system in order to allow speed up.

II. METHODOLOGY

One of the most intensive mathematical operations in a microcontroller is multiplication. The PIC16F877 microcontroller was chosen as the microcontroller of choice for the proposed parallel processing system due to the fact it does not have a multiply instruction. The PIC16F877 microcontroller is an 8-bit, -40 pin device with 33 pins of input/output (I/O) capability and 32 available instructions. The proposed system configuration will potentially enhance the device's ability to compute mathematical operations. The device's large I/O capability also makes this device an ideal choice for this investigation. The hardware configuration of the dual microcontroller parallel processing system is shown in Figure 1. In Figure 1 microcontroller 1 is assigned as the Master and microcontroller 2 is assigned as the Slave. Eight bit data is transferred from the Master to the Slave via the eight bit Data Transmit (Tx), Data Receive (Rx) bus connection. The control of this data transfer is via a two wire handshake. The operation of this protocol is as follows. The Master places the eight data bits to be transferred onto its Tx output bus and asserts its Data Ready Signal. The Slave polls its Rx input waiting for a high signal which indicates to the Slave that data from the Master is available and ready to be read. When the Slave reads a high level on its Rx handshake control line it reads the data from the eight bit Rx bus and asserts its Data Receive Acknowledge (Data Rx Ack) control line. The Master is polling its Data Transfer Acknowledge (Data Tx Ack) control signal at this time waiting for a high which indicates that the Slave has read the eight bit data on the data transfer bus. The Master then sets its Data Ready handshake line low (inactive). The Slave reads this transition on its Rx handshake line and sets its Data Rx Ack line low. The Master reads the transition on this control line on its Data Tx Ack and the data handshake protocol is complete. This is a comprehensive two wire data handshake which will require some processing time to complete, however it does ensure safe transfer of data of data from Master to Slave. There will also be occasion for the Slave to transfer data to the Master. This is achieved by reversing the function and direction of the handshake control lines and the direction of the eight bit data transfer bus. The RAM memory in Figure 1 is required to store the data necessary to assess the performance of the parallel processing system. The active low input to the RAM, /CE, is driven by the Master and is used to enable the RAM memory. The /WE input to the RAM, again driven by the Master microcontroller, is used to write data values, the number of clock cycles executed, into the RAM memory.

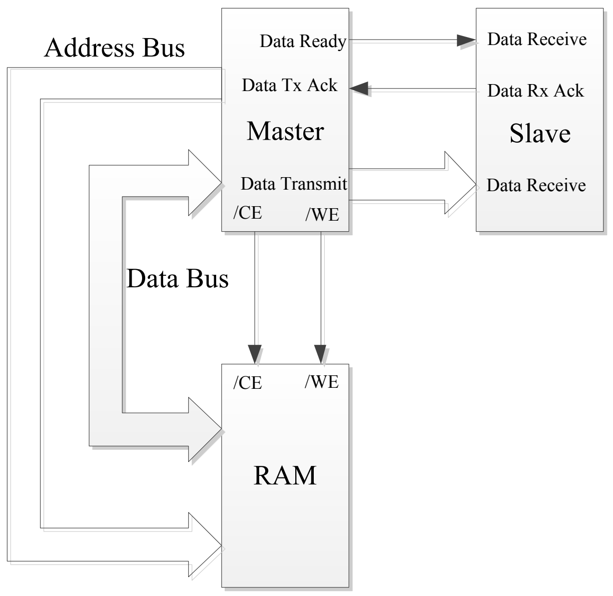

Figure 1: Block diagram of the proposed system

In order to test the performance of the parallel processing system an 8x8 bit multiply operation and a 16x16 bit multiply are performed. The multiply instruction was considered to be a good function to use as it has good potential parallel processing capability whereby the intensive multiplication calculation can be effectively shared between the two microcontrollers. The fact that 16F877 does not have a multiply instruction within its instruction set, also gave the multiply as the test calculation, greater credibility.

The eight bit multiply executed on the parallel processing is performed as follows.

The eight bit multiplicand and eight bit multiplier values, which for testing processes are generated in the Master microcontroller, are transferred to the Slave. The calculation of the multiply operation is effectively shared between Master and Slave. The Master calculates the lower partial product by determining the result of the 8 bit multiplicand multiplied by the lower nibble of the multiplier. The Slave calculates the upper partial product by determining the result of the multiplicand multiplied by the upper nibble of the multiplier. In both cases the multiply operation is performed by a process of shift and add. The two eight bit values, the partial product result calculated by the Slave, are then transferred to the Master via the two wire handshake transfer protocol. The Master then sums the two eight bit values calculated by the master with the two eight bit values calculated by the slave to determine the overall multiply result as shown in Table 1. Notice that the Slave partial product must be shifted four places to the left before completing the final addition (X, Y and Z represent single hexadecimal characters).

Table 1 Calculation of Master and Slave partial products

	8x8_multiply			
Master partial product calculation			X	X
Slave partial. product calculation	+	Y	Y	
Overall Multiply Result	Z	Z	Z	Z

The 16x16 bit multiply is completed largely in the same way as the 8x8 multiply. The 16 bit multiplicand and 16 bit multiplier are transferred to the Slave via the two wire handshake protocol as two four byte data transfers. The Master calculates the partial product of the multiply operation by multiplying the multiplicand by the lower byte of the multiplier, and the Slave calculates the partial product of the multiply by multiplying the multiplicand by the upper byte of the multiplier.

In the case of the 16x16 bit multiply the shifted multiplicand is held in three, eight bit registers, and the partial product is calculated after eight iterations of the multiplying process. The result of the Slave partial product calculation is stored in three, eight bit registers and is transferred to the Master after the calculation is completed. The Master then adds the two calculated partial products to produce the overall result.

An 8x8 multiply and 16x16 bit multiply were also performed on a single microcontroller in order to be able to quantify the performance of the parallel processing system. The number of ones and zeros in the multiplier and multiplicand will be the governing factor in the number of clock cycles required to complete the multiply calculations. The test patterns generated have been developed in order to increase, reduce and move around the number of ones and zeros within the multiplier and multiplicand in order to give a good representation of possible calculation values without the necessity to complete all of the possible calculation permutations. The 256 results from the generated test patterns will give a good representation of the performance of both the 8x8 multiplier and the 16x16 multiplier.

For the 8x8 bit multiplier, sixteen multiplicand values are multiplied by sixteen multiplier values. The multiplier is initially set at 00 and multiplied by the multiplicand starting from 00, and then through the values 03, 0C, 0F, 30, 33, 3C, 3F, C0 … to FF. The multiplier is then set 03 and multiplied through the same multiplicand values as previous. The next multiplier value is 0C, again multiplied by the same multiplicand values, until eventually the last multiplication FFxFF is completed when a total of 256 calculations will have been completed. The sixteen test pattern values have been chosen so that two adjacent '1's effectively increment through the multiplicand and also ultimately through the multiplier. All possible combinations of values with two adjacent '1's are thus included as well as all combinations with four adjacent '1's.

For the 16x16 bit multiplier, similar calculations are performed with the values being 0000, 000F, 00F0, 00FF, 0F00, 0F0F, 0FF0, 0FFF, F000 … to FFFF. Thus the last multiply calculation completed will be FFFFxFFFF when 256 multiplications will have been completed. In this case, four adjacent '1's effectively increment through multiplicand and multiplier in a similar fashion to the test patterns applied to the 8x8 multiplier.

The time duration of the 8x8 multiplier calculations are measured by TMR0, an eight bit counter within the Master microcontroller and the time duration of the 16x16 calculations are measured by TMR1, a 16 bit timer within the Master microcontroller.

III. RESULTS AND ANALYSIS

The results shown in Figure 2 for the single multiplier follow an approximate sawtooth pattern. The number of '1's in the multiplier effectively increases the execution time of the multiplication. The more the number of '1's the more clock cycles required. This is due to the implemented algorithm whereby if the multiplier bit under interrogation is a 1 then the shifted multiplicand is added to the running total accumulator and then shifted multiplicand is shifted left one bit. If the multiplier bit under interrogation is a zero the shifted multiplicand is shifted left one bit without addition. The add operation incurred by the presence of '1's in the multiplier is the cause of the increase in the number of cycles. It can be seen however that the increase in clock cycles is not linear. This non linearity is a consequence of the number of '1's in the multiplicand which can also have an impact on the number of clock cycles required to complete the multiplication. If a '1' is in the most significant bit position of the lower byte of the shifted multiplicand when it is shifted left, then this '1' bit must appear in the least significant bit of the upper byte of the shifted multiplicand after the shift left of the shifted multiplicand has been completed. This requires some additional programming which consequently increases the number of clock cycles.

The sawtooth pattern of the 8x8 multiplier from the single microcontroller shown in Figure 2 arises as result of the sequence of test patterns applied to the multiplier. As the number of clock cycles rises towards a peak, the number of '1's in the multiplicand will be steadily increasing. As the multiplicand reaches its maximum value of FF a peak will result. The next test pattern will load 00 into the multiplicand which is a reduction of eight '1's causing the number of clock cycles required to fall abruptly. This can be demonstrated in Figure 2 as the number of clock cycles rises toward the first peak. At the first peak the test pattern FFx03 is executed when 136 clock cycles are required. The next text pattern executed is 00x0C (arrowed) requiring only 117 clock cycles, an abrupt reduction in the number of clock cycles required, and thus the sawtooth pattern is generated.

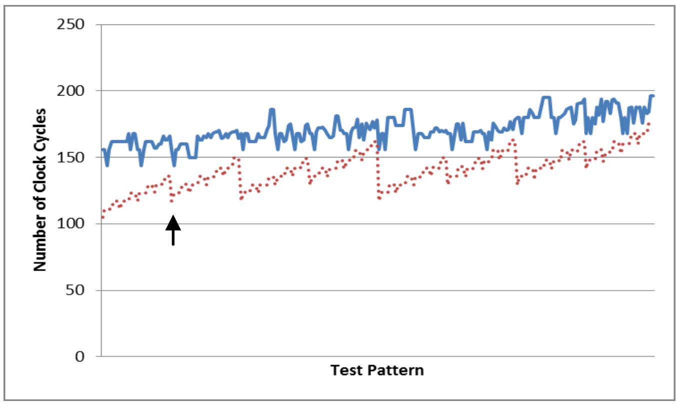

Figure 2 Number of clock cycles agiainst Test Patterns for single microcontroller 8x8 multiplier (dotted), and dual microcontroller 8x8 multiplier (solid).

It would be a natural assumption to believe that the results for the 8x8 multiplier carried out by the dual microcontroller would follow a similar pattern, however upon consultation of the results in Figure 2, this is shown not to be the case. It would also be hoped that the execution times for the dual microcontroller solution would be less than that of the single microcontroller as the dual microcontroller solution should complete the multiplication in half the time, but again from Figure 2 this is seen not to be the case. This can be explained by consideration of Table 2.

Tx Time A is the number of clock cycles required to transfer the one byte multiplicand and one byte multiplier from the Master to the Slave. The Multiply Time is the number of clock cycles required to complete the shift and add process completed simultaneously in the Master and the Slave. Tx Time B is the number of clock cycles required for the Slave to transfer the calculated two byte partial product to the Master and the Addition Time is the number of clock cycles required to add the partial product generated by the Master to the partial product generated by the Slave. The table shows that there is a steady increase in the Multiply Time as the number of ‘1’s in the multiplier and Multiplicand increases, however this increase in the number of clock cycles is relatively small from 80 clock cycles for the 00x33 (six ‘1’s in the multiplier) multiply to 107 clock cycles for the FFxFF (eight ‘1’s in the multiplier and eight ‘1’s in the multiplicand). The addition time is relatively constant at seven or eight clock cycles. The problem with the eight bit multiplier carried out by two microcontrollers is the significant amount of time to transfer the data from Master to Slave and back again which on occasion exceeds and is certainly comparable to, the number of clock cycles required to carry out the multiply. Thus, the speed up $S(n)$ for the dual microcontroller system which would have hoped to be approaching two, is actually less than one for all multiplications completed in the test. This means that the single multiplier is actually faster at carrying out a multiplication than the dual, parallel processing microcontroller system, due to the overhead of the data transfer time between the Master and the Slave.

The results shown in Figure 3 for the 16x16 multiply completed on a single microcontroller follows a similar sawtooth pattern as the 8x8 multiply for the single microcontroller shown in Figure 2. As expected, the number of clock cycles required to complete the multiplication increases as the number of ‘1’s in the multiplier and the multiplicand increases. Results for the 16x16 multiply for the dual parallel processing microcontroller system depicted in Figure 3 again shows the number of clock cycles required to complete the multiplication increases as the number of ‘1’s in the multiplicand and multiplier increase. A drop in the number of clock cycles occurs when the number of ‘0’s in the test pattern

Table 2 Component cycle times for dual microcontroller 8x8 multiplier

Multiply	00x33	C3x33	CCxCC	FFxFC	FFxFF
Tx Time A	44	50	44	50	44
Multiply Time	80	80	83	105	107
Tx Time B	31	38	28	28	37
Addition Time	7	7	7	8	8
Total time	162	175	162	191	196

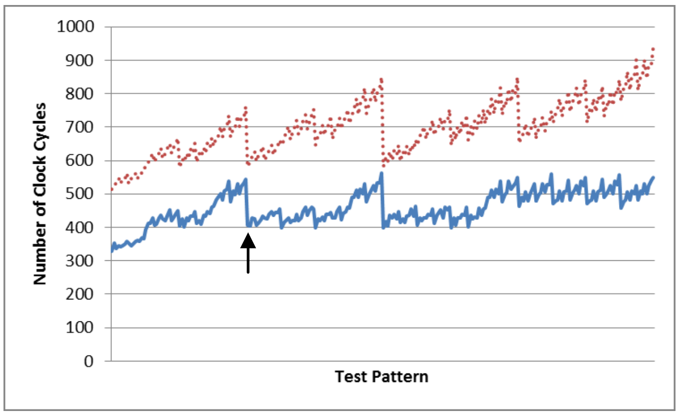

Figure 3 Number of clock cycles agiainst Test Patterns for single microcontroller 16x16 multiplier (dotted), and dual microcontroller 16x16 multiplier (solid).

increases relative to the previous test pattern e.g. FFFFx00FF requires 545 clock cycles while the next test pattern carried out is 0000x0F00 (arrowed) which requires only 407 clock cycles. The detail of the clock cycle breakdown for a number of 16x16 multiplications is shown in Table 3. The significant difference between the results shown in Table 3 compared with those in Table 2 is that the time to transfer the data between master and slave is significantly less than the multiply time. Thus the total cycles required to complete all possible multiplications by the dual microcontroller system is less than the equivalent multiplications carried out by the single microcontroller. This yields the speed up performance shown in Figure 4. From Figure 4 it can be seen that the average speed up factor for the dual microcontroller carrying out a 16x16 multiply is 1.5.

Table 3 Component cycle times for dual microcontroller 16x16 multiplier

Multiply	FF00x 00F0	F0F0x 0F0F	0F0Fx F0FF	FFFFx F0F0	FFFFx FFFF
Tx Time A	64	64	64	64	64
Multiply Time	275	313	377	329	425
Tx Time B	58	61	58	46	46
Addition Time	10	10	14	15	15
Total Time	407	448	513	454	550

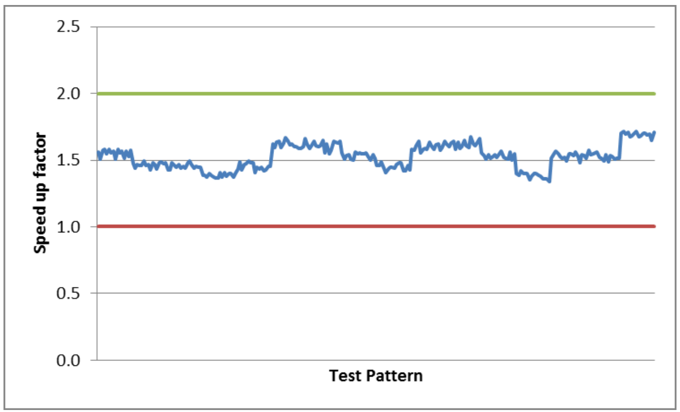

Figure 4 Speed-up against Test Patterns for 16x16 dual microcontroller

For comparison, speed up factor of 2 is highlighted in Figure 4 to indicate the maximum possible speed up for two microcontrollers operating in parallel and speed up factor of 1 is highlighted to show the speed up for a single microcontroller.

For an algorithm with good parallel processing potential such as the multiply operation where the majority of the calculation can be shared equally between *n* microcontrollers, generally from equation (1), $t_p(n)$ can be approximated as:

$$t_p(n) \approx \frac{t_p(1)}{n} + t_{transfer} \qquad (3)$$

Where n is the number of microcontrollers and $\frac{t_p(1)}{n}$ assumes that the algorithm can be completed in the time required to complete the operation by one microcontroller divided by the number of microcontrollers in the system *n.* In this case where n=2, the two microcontrollers complete the mathematical operation in half the time required to complete the operation by a single microcontroller and $t_{transfer}$ is the time required to transfer data between Master and Slave, and Slave and Master.

By substituting into (1) it can be shown that, for a dual microprocessor system, to achieve a speed-up factor of $S(n) = 1$, $t_p(1)$ is required to be completed in twice the time required to complete the data transfer, $t_{transfer}$. In order to achieve a moderate speed up factor of 1.5, $t_p(1)$ is required to be completed in six times the time required to complete the data transfer.

IV. CONCLUSION

An 8x8 bit multiplication and a 16x16 bit multiplication have been implemented on a single microcontroller and a dual microcontroller operating as a parallel processing system. Results have shown that for the 8x8 multiplication the speed-up factor $S(n)$ was found to be less than 1. This is due to the relatively large time required to transfer data between the two microcontrollers compared to the time required to complete the multiply operation. The time effectively saved by completing the multiplication using two microcontrollers is lost with the time required to transfer the data between the two microcontrollers . For the 16 x 16 bit multiplication a moderate average speed up factor of 1.5 was achieved due to the fact that the transfer time is very much less than the multiply time. The overall deciding factor as to whether speed-up can be achieved was found to be dependent upon the time required to transfer data between the two microcontrollers. A formula for an algorithm with good parallel processing capability has been developed to calculate whether speed-up can be achieved. This formula indicates that to achieve a speed-up factor of 1, the time required to complete the calculation on a single processor must be twice the time required to transfer the data between two microcontrollers. To achieve a good speed-up factor of 1.8 for a two microcontroller parallel processing system, from the developed equation, the time required to complete the operation on a single microcontroller must be ten times that required to transfer the data between the two microcontrollers. To significantly improve the speed up time for the parallel processing system the challenge will be to reduce the transfer time between the microcontrollers which will provide some focus for future work.

REFERENCES

[1] F. Gebali, Algorithms and Parallel Computing: John Wiley & Sons, 2011.

[2] Y.-l. Hu, J.-l. Cao, F. Ran, and Z.-j. Liang, "Design of a high performance microcontroller," in High Density Microsystem Design and Packaging and Component Failure Analysis, 2004. HDP '04. Proceeding of the Sixth IEEE CPMT Conference on, 2004, pp. 25-28.

[3] J. Von Neumann, "First draft report on the EDVAC," 1945.

[4] "PIC16F877 Datasheet," Microchip Technology. Accessed on 2/2/2012

[5] T. Sakamoto and T. Hase, "Software JPEG for a 32-bit MCU with dual issue," Consumer Electronics, IEEE Transactions on , vol.44, no.4, pp.1334-1341, Nov 1998

[6] O. Maslennikov, J. Shevtshenko, and A. Sergyienko, "Configurable microcontroller array," Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on , vol., no., pp. 47- 49, 2002

[7] T. Schubert and B. Becker, "Lemma exchange in a microcontroller based parallel SAT solver," VLSI, 2005. Proceedings. IEEE Computer Society Annual Symposium on , vol., no., pp. 142- 147, 11-12 May 2005

[8] M. Bin, "Dual-microcontroller based GPRS data transmission control system design," Information Technology and Artificial Intelligence Conference (ITAIC), 2011 6th IEEE Joint International , vol.1, no., pp.101-104, 20-22 Aug. 2011

[9] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in AFIPS Conference, New Jersey, 1967, pp. 483-485.