A Novel High-Speed Configurable Viterbi Decoder for Broadband Access

Mohammed Benaissa

Department of Electronic and Electrical Engineering, The University of Sheffield, Mappin Street, Sheffield S1 3JD, UK
Email: m.benaissa@sheffield.ac.uk

Yiqun Zhu

Department of Electronic and Electrical Engineering, The University of Sheffield, Mappin Street, Sheffield S1 3JD, UK
Email: elp99yz@sheffield.ac.uk

A novel design and implementation of an online reconfigurable Viterbi decoder is proposed, based on an area-efficient add-compare-select (ACS) architecture, in which the constraint length and traceback depth can be dynamically reconfigured. A design-space exploration to trade off decoding capability, area, and decoding speed has been performed, from which the maximum level of pipelining against the number of ACS units to be used has been determined while maintaining an in-place path metric updating. An example design with constraint lengths from 7 to 10 and a 5-level ACS pipelining has been successfully implemented on a Xilinx Virtex FPGA device. FPGA implementation results, in terms of decoding speed, resource usage, and BER, have been obtained using a tailored testbench. These confirmed the functionality and the expected higher speeds and lower resources.

Keywords and phrases: pipelining, configurable, ACS, area-efficient architecture, design-space exploration, schedule.

1. INTRODUCTION

Overcoming the variable deterioration in the reliability of a broadband communication channel in real time is a critical issue. That is why channel-coding techniques such as convolutional codes represent an important part of any broadband communication system. For example, DSL, WLAN, and 3G standards all require variations of convolutional coding with differing coding performance (constraint length and code rate) at differing data rates and therefore require differing decoding performance, usually using Viterbi decoding [1]. Therefore, from the viewpoint of channel-coding techniques, this demands both high decoding speed and variable decoding capability to match the channel conditions. Furthermore, it is becoming increasingly important to develop hardware implementations that can operate over a range of standards and can support multiple networks without redesign. Hence both hardware performance and flexibility are crucial. This requires high-speed, low-power dynamically reconfigurable forward error control coding dedicated hardware architectures that can operate within a range of channel conditions under a number of speed/power performance constraints at different time intervals.

Designing and implementing such architectures is a challenging problem for large constraint lengths Viterbi decoders since decoding capability and decoding complexity are closely related to the constraint length used. A larger constraint length can offer a higher decoding capability but at the expense of a higher decoder complexity, often in terms of a cost function of resource usage versus decoding delay versus decoding capability, depending on the specific hardware architecture adopted. A useful Viterbi decoder architecture will therefore offer the flexibility to trade off the parameters of this cost function with reasonable performance. This requires architectural level decisions to allow optimum resource sharing and maximum pipelining to achieve a practical compromise between resource usage and decoding performance for a range of constraint lengths. Such architectural decisions would range from state-parallel to state-serial architectures. On the one hand, a state-parallel architecture, in which the number of ACSs is equal to the number of states and all ACSs operate in parallel, can offer high decoding speed, which only depends on the computation delay of the ACS feedback loop. However, the hardware complexity increases exponentially with the constraint length of the convolutional codes and this makes these architectures often unsuitable for applications requiring codes with large constraint lengths such as 3G (constraint length 9). On the other hand, in a state-serial architecture (sometimes referred to as software solutions), all states share one ACS; although flexible,

such architecture would result in a huge decoding delay for large constraint lengths, hence limited throughput to suit most broadband applications. An area-efficient/foldable architecture as proposed in [2, 3, 4, 5] uses more than one ACS. The number of ACSs to be used depends on the requirement of resource usage, and as such this class of architectures is attractive for a configurable implementation solution for large constraint lengths without excessive penalties in terms of resource usage. However, their speed performance suffers when the ratio of number of states to number of ACS units increases. Therefore, such architectures would only be possible for broadband access performance if their design space is explored in terms of maximum speedup (pipelining) versus number of ACS units (area) versus constraint length (decoding capability).

In this paper, we investigate the design space for area-efficient Viterbi decoders and develop an online reconfigurable architecture that will support a range of constraint lengths without an excessive loss of speed performance.

A scheduling program is used to systematically determine the maximum level of pipelining (speedup) that can be applied to the decoder in an area-efficient/foldable architecture with in-place path metric updating [6]. This enables the exploration of the trade-off of decoding speed (throughput) versus area (number of ACS units) for a range of constraint lengths.

This exploration is undertaken for a range of constraint lengths from 7 to 10 selected to cover many broadband access applications and also this range is challenging enough in terms of complexity to validate the design approach adopted. The optimum solution in terms of throughput versus area versus decoding capability (which is limited here by constraints 7 to 10) yielded a maximum level of pipelining of 5 levels for an area-efficient architecture with 8 ACS units using in-place path metric updating. This gives a speedup of 5 times on designs using a similar area-efficient/foldable architecture and achieves 5/8 the speed of a state-parallel architecture. The speed/throughput of course is determined by the requirements of the lowest constraint length, in this case, 7. In addition to the in-place updating, pipelining also enables reduction in path metric memory by allowing lower bit resolution for the computations.

The design is then implemented on a Virtex FPGA and tested using a developed hardware testbench. Actual hardware performance figures and BER curves are obtained to confirm the functionality and performance improvements.

It is important to note that Viterbi decoders have been widely investigated and implementations of configurable decoders have been reported in many papers. For example, [7] implemented an adaptive Viterbi decoder (AVD) based on reconfigurable processor board (RCPB), in which the constraint lengths can be reconfigured from 7 to 15. The AVD is specifically designed for an FPGA platform by using the features of FPGA configuration, so it is not suitable for the application where instant online reconfiguration is required due to the very low-speed FPGA configuration. In [8], a reconfigurable Viterbi decoder architecture, the constraint lengths which can be reconfigured from 3 up to 7, was proposed by adopting a state-parallel ACS module. Because the hardware complexity of state-parallel ACS architectures is exponentially proportional to the constraint length, this approach is not suitable for large constraint lengths.

To our knowledge, the approach adopted in this paper, the level of performance improvements, and the trade-offs achieved have not been reported before.

The paper is organised as follows. A brief design-space exploration is given in Section 2. The architecture of a configurable Viterbi decoder example is described in Section 3. FPGA implementations and performance comparisons based on the FPGA prototype are given in Section 4. Comparisons and conclusions are presented in Sections 5 and 6, respectively.

2. DESIGN-SPACE EXPLORATION FOR AREA-EFFICIENT ARCHITECTURES

As already mentioned in the introduction, the trade-off area versus speed versus decoding capability is crucial in a reconfigurable area-efficient/foldable Viterbi architecture. In our case, decoding capability corresponds to the constraint length, area corresponds to the number of ACS units used, and speed corresponds to the throughput achieved, which can be assimilated in this case to the number of pipeline levels that can be inserted in the ACS feedback loop.

A software program was written to explore this 3D design space in order to determine an optimum solution while maintaining a standard resource saving techniques known as in-place path metric update. The results are shown in Table 1.

A number of interesting observations can be made at this stage. The first column of course refers to a state-parallel architecture ($N = P$), which achieves the best speed/throughput that we note as F (Mbps), for example. The second and third columns show that halving the number of ACS units ($P = N/2$) is the worst solution as it does not give any speedup (pipelining) advantage. In fact we can achieve the same throughput rate of $F/2$ by using a 2-level pipelining of the ACS feedback loop on a quarter of the number of ACS units ($P = N/4$). This corresponds to a speedup by a factor of 2. The extreme case of the last column shows that a throughput rate of $5F/8$ can, in theory, be maintained on a number of ACS units $P = N/32$ as long as we can insert 20 levels of pipelining. Of course pipeline balancing is a critical issue in this case and adopting such a solution in practice would not be advisable.

The optimum solution from a practical hardware implementation viewpoint is the fourth column which corresponds to using a number of ACS units $P = N/8$. This gives a

TABLE 1: 3D design exploration of area-efficient Viterbi decoders.

States/ACS units (N/P)	1	2	4	8	16	32
ACS pipeline levels	1	1	2	5	10	20
Throughput/speed (Mbps)	F	$F/2$	$F/2$	$5F/8$	$5F/8$	$5F/8$

TABLE 2: 120 2-bit index data arrangement in each ROM (128×2).

Constraint length (K)	—	7	8	9	10
ROM address	0–7	8–15	16–31	32–63	64–127

FIGURE 1: The diagram of BF unit.

5 times speedup by inserting judiciously 5 levels of pipelining in the ACS feedback loop; often some careful timing analysis is required here. For a configurable design for constraint lengths from 7 to 10, this optimum solution translates to $64/8 = 8$ ACS units with 5 levels of pipelining. The maximum throughput is governed by the requirements of constraint length 7.

The next section explains in detail the issues involved in the context of a design example.

3. CONFIGURABLE VITERBI DECODER ARCHITECTURE

A reconfigurable Viterbi decoder, which is based on an area-efficient ACS architecture, is composed of a branch metric (BM) module, an ACS module, a best-state module, and a traceback module.

3.1. BM module

The BM module is to generate the BMs [9] for the proper butterfly (BF) units in the ACS module at the proper time unit. For our configurable Viterbi decoder, considering the whole range of constraint lengths 7, 8, 9, and 10, there are 480 possible different BF operations, in which 32, 64, 128, and 256 BF operations are needed for constraint lengths 7, 8, 9, and 10, respectively. Each different BF operation needs 2-bit index data to identify its corresponding BM from 4 possible BMs. On the other hand, all the 480 BF operations are equally distributed for four available BF units, each BF unit is responsible for 120 possible different BF operations. As a result, 120 2-bit index data are required for each BF unit to select proper BMs for 120 possible BF operations. Hence the BM module can be configured to provide BMs for one specific constraint length from the constraint lengths from 7 to 10.

To be easily implemented, a ROM (128×2) is used to store the 120 2-bit index data needed for each BF unit. For each ROM, the 120 2-bit index data are arranged as shown in Table 2 as this allows for easy hardware implementation. The first 8 addresses (0 to 7) are not used, and then 8 addresses (8 to 15), 16 addresses (16 to 31), 32 addresses (32 to 63), and 64 addresses (64 to 127) are used for constraint lengths 7, 8, 9, and 10, respectively.

3.2. ACS module

In the proposed architecture, this module is the most critical part, in which a novel ACS pipeline scheme is implemented to achieve higher ACS computation speed. To better describe the ACS pipeline scheme, we consider the case of constraint 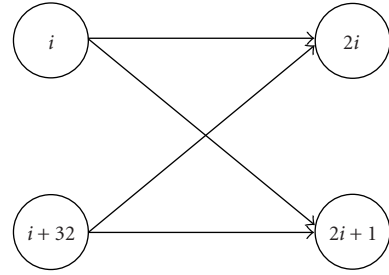length 7, so the number of states is 64. Assume that the number of available ACS units is 8. The key feature of the proposed ACS pipeline scheme is to speed up ACS operations by inserting the maximum number of ACS pipeline levels.

For the purpose of simplification, BF units, rather than ACS units, are used to explain the proposed scheme. The diagram of BF unit is illustrated in Figure 1. Each BF unit consists of two ACS units that share the same input and output states. More specifically, for each BF, the path metrics for two current states are obtained from the current BMs and the path metrics of two previous states, which lead to current states by executing two ACS operations.

The overall architecture of the ACS module is shown in Figure 2. BF0, BF1, BF2, and BF3 are BF units. There are 4 BF units, which make up 8 ACS units as used in our area-efficient ACS module. Switch0 and Switch1 are 4×4 switches, the function of which, as given in Table 3, is to permute the path metric network in such a way that the global routing network can be localized by these regular bus-switch components. Different from [10], in order to have an identical simplified architecture for all BF units, a 4×4 switch is used instead of two 2×2 switches. DpRAM0 to DpRAM7 are dual port RAMs used for path metric memory. With in-place path metric updating, the required path metric memory size is equal to the number of path metrics, which is the same as the number of states (there are 64 states for our case). So the depth of each path metric memory DpRAM is 8.

The initial arrangement of all the 64 path metrics in the path metric memory is given at iteration 0 in Table 4, in which the state number is used to denote the corresponding path metric. For instance, the path metric of state 2D is assigned into dual-port memory DpRAM1 at address 5, and will be the output to BF0 as PmIn01 for ACS computation. Following the architecture of the ACS module shown in Figure 2, with proper selection control as shown in Table 3, the state distribution at iteration 1 can be obtained from iteration 0 after 8 cycles by executing in-place path metric updating. Each iteration takes 8 cycles and the initial arrangement of the state of path metrics in DpRAM is re-established after 6 iterations in terms of the property of in-place path metric updating technique [6]. Only iterations 0 and 1 are given in Table 4, in which we can see that due to in-place path metric updating, the path metric distributions are different between iterations 0 and 1.
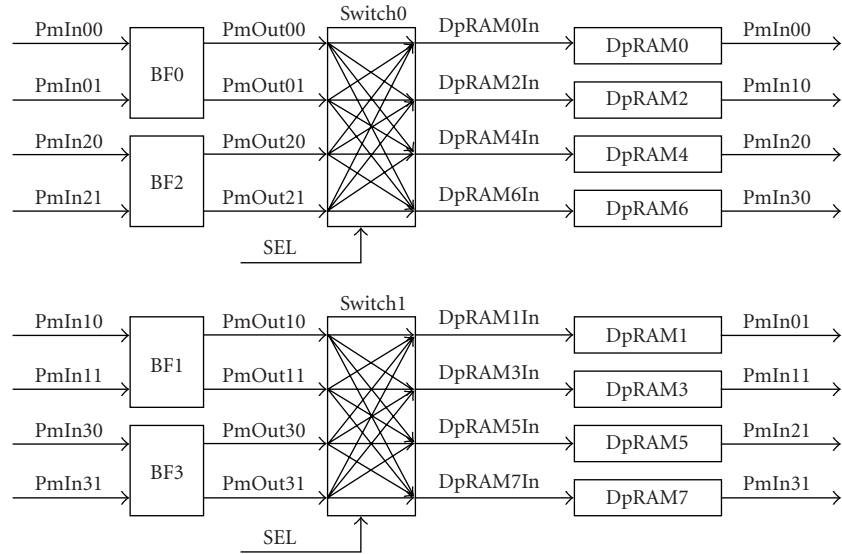
FIGURE 2: The architecture of the ACS module.

TABLE 3: Selection control for Switch0 and Switch1.

SEL	00	01	10	11
DpRAM0In	PmOut00	PmOut01	PmOut20	PmOut21
DpRAM2In	PmOut20	PmOut21	PmOut00	PmOut01
DpRAM4In	PmOut01	PmOut00	PmOut21	PmOut20
DpRAM6In	PmOut21	PmOut20	PmOut01	PmOut00
DpRAM1In	PmOut10	PmOut11	PmOut30	PmOut31
DpRAM3In	PmOut30	PmOut31	PmOut10	PmOut11
DpRAM5In	PmOut11	PmOut10	PmOut31	PmOut30
DpRAM7In	PmOut31	PmOut30	PmOut11	PmOut10

TABLE 4: State arrangement and in-place path metric updating.

Iteration		0							
Address (DpRAM0–7)		0	1	2	3	4	5	6	7
BF0	DpRAM0	00	04	02	06	09	0D	0B	0F
	DpRAM1	20	24	22	26	29	2D	2B	2F
BF1	DpRAM2	10	14	12	16	19	1D	1B	1F
	DpRAM3	30	34	32	36	39	3D	3B	3F
BF2	DpRAM4	08	0C	0A	0E	01	05	03	07
	DpRAM5	28	2C	2A	2E	21	25	23	27
BF3	DpRAM6	18	1C	1A	1E	11	15	13	17
	DpRAM7	38	3C	3A	3E	31	35	33	37
Iteration		1							
Address (DpRAM0–7)		0	1	2	3	4	5	6	7
BF0	DpRAM0	00	09	04	0D	02	0B	06	0F
	DpRAM1	20	29	24	2D	22	2B	26	2F
BF1	DpRAM2	10	19	14	1D	12	1B	16	1F
	DpRAM3	30	39	34	3D	32	3B	36	3F
BF2	DpRAM4	01	08	05	0C	03	0A	07	0E
	DpRAM5	21	28	25	2C	23	2A	27	2E
BF3	DpRAM6	11	18	15	1C	13	1A	17	1E
	DpRAM7	31	38	35	3C	33	3A	37	3E

Obviously, address scrambling is required for in-place path metric updating to be executed, in other words, address scrambling is used to schedule the right path metric into the right cycle in order for the same set of path metrics to be read into BF units for ACS operation at the same cycles of any iteration. There are many different address scrambling methods, all of which can meet the requirements of in-place path metric updating. However, besides in-place path metric updating scheme, another requirement of address scrambling is that the maximum number of pipeline levels can be obtained without any impact of in-place path metric updating. For further discussion, we consider two specific address scrambling methods as shown in Table 5 in which only the first two iterations are given.

For Address scrambling 1, for any path metric memory, the path metric is read from address i at cycle i of iteration 0, where i is from 0 to 7. At iteration 1, for path metric memory, DpRAM0 to DpRAM3, the path metrics are read from addresses 0, 2, 4, 6, 1, 3, 5, and 7 at cycles 0, 1, 2, 3, 4, 5, 6, and 7, respectively, while for DpRAM4 to DpRAM7, the path

metrics are read from addresses 1, 3, 5, 7, 0, 2, 4, and 6 at cycles 0, 1, 2, 3, 4, 5, 6, and 7, respectively. By address scrambling, at any iteration, the same path metrics will be read out at the same cycles as in the first iteration. For example, at cycle 4 of any iteration, the path metrics of state 09, 29, 19, 39, 01, 21, 11, and 31 must be read from the path metric memory into 4 BF units, BF0, BF1, BF2, and BF3. After the multiplexing of the two switches, Switch0 and Switch1, the output path

TABLE 5: Two address scrambling methods of path metric memory.

Cycle	0	1	2	3	4	5	6	7
Address scrambling 1								
SEL	00	01	00	01	10	11	10	11
Iteration	0							
Address (DpRAM0–3)	0	1	2	3	4	5	6	7
Address (DpRAM4–7)	0	1	2	3	4	5	6	7
Iteration	1							
Address (DpRAM0–3)	0	2	4	6	1	3	5	7
Address (DpRAM4–7)	1	3	5	7	0	2	4	6
Address scrambling 2								
SEL	00	01	00	10	01	11	10	11
Iteration	0							
Address (DpRAM0–3)	0	1	2	4	3	5	6	7
Address (DpRAM4–7)	0	1	2	4	3	5	6	7
Iteration	1							
Address (DpRAM0–3)	0	2	4	1	6	3	5	7
Address (DpRAM4–7)	1	3	5	0	7	2	4	6

TABLE 6: The allowed cycles for ACS for address scrambling 1.

Cycle	0	1	2	3	4	5	6	7
The allowed cycles	8	7	7	6	6	5	5	4

TABLE 7: The allowed cycles for ACS for address scrambling 2.

Cycle	0	1	2	3	4	5	6	7
The allowed cycles	8	7	7	7	5	5	6	5

metrics of state 02, 22, 12, 32, 03, 23, 13, and 33 will be written back to the path metric memory with the same address. From Tables 3 and 4, we can see that the output path metrics of state 02, 22, 12, and 32 will not be read until 6 cycles later, while the output path metrics of state 03, 23, 13, and 33 will not be read until 10 cycles later. Therefore, 6 cycles can be allowed for the ACS computations of the fourth cycle path metrics. In other words, 6 cycles can be available for the ACS computations of the path metrics read out at cycle 4 without any impacts on in-place path metric updating. Likewise, at any other cycle, the number of cycles allowed from the corresponding ACS computation can be worked out, which is given in Table 6.

From the point of view of the entire ACS module, with address scrambling 1, 4 cycles are available for the ACS computation, in other words, 4 pipeline levels can be inserted into ACS feedback loop to speed up ACS computation.

By applying the same method to address scrambling 2, which is obtained from the address scrambling 1 by swapping the addresses between cycles 3 and 4, the corresponding allowed cycles for ACS are obtained as in Table 7. As a result of address scrambling 2, 5 pipeline levels can be available for ACS operations.

TABLE 8: The maximum pipeline levels for constraint lengths from 7 to 10 with the usage of 8 ACS units.

Constraint length (K)	7	8	9	10
ACS pipeline levels	5	10	20	40

From the above discussion, for our area-efficient ACS module with constraint length 7 and the area saving requirement of 8 ACS units, at least 5 pipeline levels can be introduced for the ACS operation. However, by using exhaustive computer search, we found that 5 is the maximum number of pipeline levels which can be introduced for the above area-efficient ACS module.

With the usage of 8 ACS units, the maximum number of ACS pipeline levels can be worked out for constraint lengths from 7 to 10 as shown in Table 8.

Therefore, in order to implement our ACS module, in which constraint length can be reconfigurable from 7 to 10 with the restriction of 8 ACS units, 5 ACS pipeline levels can be inserted into ACS feedback loop.

To reduce the delay of the ACS computational loop, two's complement arithmetic [11] is normally used for implicit renormalization of the path metrics. Furthermore, in order to enable modulo normalization of the path metrics, according to [12, 13], the minimum resolution of the path metrics is given by

$$
\begin{aligned}
\Delta_{\max} &= \lambda_{\max} \log_2 N, \\
\Gamma_{\text{bits}} &= \lceil \log_2 (\Delta_{\max} + k\lambda_{\max}) \rceil + 1,
\end{aligned}
\tag{1}
$$

where N is the number of states, $\lambda_{\max}$ is maximum BM, and k is 1 and 2 for radix-2 ACS and radix-4 ACS, respectively. Hence, for a maximum constraint length 10 and radix-2 ACS with 3-bit quantisation, $N = 512$, $k = 1$, and $\lambda_{\max} = 14$; thus 1 gives a minimum resolution of the path metrics of 9 bits. In other words, at least 9-bit data width is required for path metric memory in order to use modulo normalization for the path metrics. However, in our reconfigurable Viterbi decoder, the 5-level ACS pipeline scheme allows a modified variable shift path metric normalization [12] and saturation protection circuits to be inserted into the ACS feedback loop in a pipeline fashion. This allows even lower resolution to be used for the path metric without decoding performance loss. The modified variable shift path metric normalization is realized by subtracting a constant value from all path metrics, if all path metrics is greater than this constant value, rather than subtracting the minimum path metric from all path metrics. Hence, no operation of minimum path metric selection is required in our modified variable shift path metric normalization. Saturation protection circuit, which is used to avoid catastrophic overflow, is implemented by setting the maximum value for any overflow path metrics. With our modified variable shift path metric normalization and saturation protection scheme, a 6-bit path metric is sufficient for the path metric computation in the proposed

reconfigurable Viterbi decoder, without suffering from a decoding performance penalty. Therefore, 33% reduction of path metric memory usage has been achieved, compared with the case of modulo normalization of the path metrics. In [5], a 12-bit path metric was used for adequate resolution, however, with path metric rescaling and saturation protection, and the 6-bit path metric was used for the path metric computation in the proposed configurable Viterbi decoder without suffering from a decoding performance penalty. Therefore, another 50% reduction of path metric memory usage has been achieved compared with the case of [5].

3.3. Best-state module

There are two solutions of traceback in a Viterbi decoder, best state and fixed state. In a best-state solution, the best-state survivor path is found for traceback operation, while in a fixed-state solution the survivor path of any state, usually state 0, is used for tracing back. An in-depth discussion of decoding performance for best-state and fixed-state solutions has been addressed in [14]. It is shown that, for comparable performance, the traceback depth of the fixed-state solution is as roughly twice as that of the best-state solution. As we know, the size of the survivor memory is proportional to the traceback depth, and a larger traceback depth results in more memory usage. Therefore, the survivor memory usage of a fixed-state solution can be twice that of a best-state solution. Generally, a fixed-state decoding is only employed when it is expensive to find the best state such as in the case of a state-parallel architecture with a large constraint length. For our reconfigurable Viterbi decoder, because only 8 ACS are in parallel, only 7 units compare-select (CS) are used to pick out the best state in which only a 3-cycle extra initial delay is introduced. The best-state module consists of 7 CS units working in pipeline to find the best state for the traceback module to execute the best-state traceback. Therefore, the hardware overhead for the best-state solution is significantly low.

3.4. Traceback module

In configurable traceback module, a dual-port RAM-based survivor memory is used to perform the traceback operation. Considering 8 ACS units in parallel, each ACS unit outputs one survivor information bit and 8-bit dual-port RAM data width is used to simplify interfacing between survivor memory and 8 parallel ACS units. In order for the ACS operations to be time-efficient which demands that no ACS be idle at any time, traceback must be executed in such a way that no overflow will take place for the 8-bit survivor data stream from the ACS module. In other words, traceback module and ACS module must operate in a pipeline fashion at the same throughput rate. To be a time-efficient implementation, for our reconfigurable Viterbi decoder, the overall throughput rates have to be 1/8, 1/16, 1/32, and 1/64 bit/cycle for constraint lengths 7, 8, 9, and 10 because all states are scheduled into 8, 16, 32, and 64 cycles for constraint lengths 7, 8, 9, and 10, respectively.

TABLE 9: Time-efficient schedule for one traceback.

Constraint length	ACS (cycles)	Traceback (cycles)	Decoded bits
7		$2(\text{TB}^a + 15)$	16
8	128	$2(\text{TB} + 7)$	8
9		$2(\text{TB} + 3)$	4
10		$2(\text{TB} + 1)$	2

[a]TB is traceback depth.

We consider the case of constraint length 7 to figure out how to design a configurable traceback module to meet the overall throughput rate (1/8 bit/cycle). Generally, a traceback depth of five times constraint length is needed for the best-state traceback, and hence for constraint length 7, the required traceback depth is 35. Furthermore, in order to match the high-speed clock of the area-efficient ACS module, traceback module needs to be speeded up by scheduling 2 cycles into each traceback step. Therefore, at least 70 cycles are required to finish one traceback operation. It is scheduled in our reconfigurable Viterbi decoder that one traceback operation is executed for every 16 iterations of ACS operation. Because each iteration contains 8 cycles for constraint length 7, 128 cycles are available for one traceback operation, while 100 cycles, which is calculated from $(35 + 15) \times 2$, are needed to retrieve 16 decoded bits at each traceback operation. In this way, time-efficient decoding can be achieved since the number of cycles needed for each traceback operation is less than that of 16 iterations. Obviously, if it is highly desirable to minimise the initial decoding delay, we can schedule one traceback operation every 12 iterations. This also meets the requirement of a time-efficient implementation as the number of cycles for 12 ACS iterations, 12×8, is still greater than $(35 + 11) \times 2$ cycles which are needed to retrieve 12 decoded bits. The only drawback is a more complicated hardware architecture because 12 is not a value with the form of 2^n. By using the same method, time-efficient traceback schedule can be worked out as in Table 9.

To work out the requirement of a survivor memory size for our configurable Viterbi decoder, we have to consider the largest survivor memory usage which should occur at constraint length 10. Because one traceback operation is scheduled every 16 ACS iterations and the traceback depth is required not to be less than 50 for constraint length 10, $50 \times 64 \times 8$ bits are needed to reserve for 50 traceback steps to retrieve 2 decoded bits which take 102 cycles to finish the traceback operation. To achieve nonstop ACS operation, an extra 102×8 bits are needed to buffer the new survivor data from the ACS module during the traceback operation. Therefore, the overall memory required is $50 \times 64 \times 8 + 102 \times 8$ bits equaling to 3302×8 bits. After rounding up to binary border, we use a dual-port RAM (4096×8) as survivor memory.

It can be calculated from Table 9 that the maximum traceback depths are 49, 57, 61, and 63 for constraint lengths 7, 8, 9, and 10, respectively. For our FPGA prototype, due to

TABLE 10: Data format in survivor memory for constraint length 7.

State		Address							
		0	1	2	3	4	5	6	7
	Bit0	00	08	04	0C	12	1A	16	1E
	Bit1	01	09	05	0D	13	1B	17	1F
	Bit2	20	28	24	2C	32	3A	36	3E
Data	Bit3	21	29	25	2D	33	3B	37	3F
	Bit4	10	18	14	1C	02	0A	06	0E
	Bit5	11	19	15	1D	03	0B	07	0F
	Bit6	30	38	34	3C	22	2A	26	2E
	Bit7	31	39	35	3D	23	2B	27	2F

the survivor memory restriction (4096×8), the maximum traceback depth is 62 rather than 63 for constraint length 10.

Before going into the details of the architecture of the configurable traceback SP module, we start with the data format in survivor memory because the traceback logic is decided by the survivor data format in the survivor memory. The input data bus of DpRAM is connected to the survivor data that outputted from BF units in ACS module. From Tables 4 and 5, we know that, in area-efficient ACS module, addresses are swapped between cycles 3 and 4 to maximise the speed of ACS computation by inserting 5 pipeline levels into ACS loop. In order to simplify the hardware architecture of the traceback operation, address exchange between cycles 3 and 4, which cancels the address-swapping operation in address scrambling in Table 5, is employed before writing into survivor memory DpRAM.

To better explain the traceback logic of the configurable traceback SP module, we start by considering constraint length 7. Survivor data generated in each ACS iteration are 8×8 bits which occupy 8 address entries in survivor memory, and survivor memory receives survivor data for ACS module iteration by iteration and stores the survivor data one iteration after another. As we know, a 12-bit address is required to access all data in DpRAM (4096×8). Obviously, the low 3-bit address is used to access data within one iteration and the high 9-bit address is used to identify iteration number. Table 10 shows the resulting survivor data arrangement in DpRAM. Because the data format is the same for any iteration, Table 10 only gives the data arrangement for one iteration.

Let I be a 9-bit iteration number, let C be the low 3-bit address of the 12-bit survivor memory address, and let R be 3-bit index of 8-bit data in survivor memory. So any survivor bit in survivor memory can be identified by I, C, and R. In addition, let V be the survivor bit value with the corresponding I, C, and R. In order for traceback logic to be clearly described, I, C, R, and V are packed together and are called traceback packet in Figure 3.

Obviously, with the current traceback packet information (I, C, R, and V), the previous traceback packet can be obtained from the trellis diagram of Viterbi algorithm. By
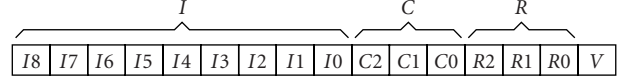

FIGURE 3: Traceback packet for constraint length 7.

checking all states, traceback formulas can be deduced as

$$R2_{\mathrm{prv}}R1_{\mathrm{prv}}R0_{\mathrm{prv}} = (R1_{\mathrm{cur}} \oplus C1_{\mathrm{cur}})VC2_{\mathrm{cur}}, \quad (2)$$

$$C2_{\mathrm{prv}}C1_{\mathrm{prv}}C0_{\mathrm{prv}} = C1_{\mathrm{cur}}C0_{\mathrm{cur}}(R2_{\mathrm{cur}} \oplus C2_{\mathrm{cur}}), \quad (3)$$

$$I_{\mathrm{prv}} = I_{\mathrm{cur}} - 1, \quad (4)$$

where the subscripts prv and cur denote the previous and current traceback steps.

Equation (4) is quite obvious because the iteration is simply updated by reducing one for each traceback step. Using an example to verify (2) and (3) assuming that the current state is 03 and the corresponding survivor bit value is "1," it can be seen from Table 10 that the corresponding current R and C are "101" and "100," respectively. Using (2) and (3), the corresponding previous R and C can be calculated as follows:

$$
\begin{aligned}
R2_{\mathrm{prv}}R1_{\mathrm{prv}}R0_{\mathrm{prv}} &= (R1_{\mathrm{cur}} \oplus C1_{\mathrm{cur}})VC2_{\mathrm{cur}} \\
&= (0 \oplus 0)11 = 011, \\
C2_{\mathrm{prv}}C1_{\mathrm{prv}}C0_{\mathrm{prv}} &= C1_{\mathrm{cur}}C0_{\mathrm{cur}}(R2_{\mathrm{cur}} \oplus C2_{\mathrm{cur}}) \\
&= 00(1 \oplus 1) = 000.
\end{aligned}
\quad (5)
$$

So the corresponding previous state is 21. On the other hand, it can be seen from the trellis diagram of Viterbi algorithm that, with survivor bit value 1, the state previous to state 03 is state 21. It is the same as that in (2) and (3).

Therefore, (2), (3), and (4) completely govern the traceback operation for constraint length 7. By using the same method, the traceback formulas for constraint lengths 8, 9, and 10 can be deduced as (6) to (12). Figure 4 shows the corresponding traceback packets for constraint lengths 8, 9, and 10.

For constraint length 8,

$$R2_{\mathrm{prv}}R1_{\mathrm{prv}}R0_{\mathrm{prv}} = (R1_{\mathrm{cur}} \oplus C2_{\mathrm{cur}})VC3_{\mathrm{cur}}, \quad (6)$$

$$
\begin{aligned}
C3_{\mathrm{prv}}&C2_{\mathrm{prv}}C1_{\mathrm{prv}}C0_{\mathrm{prv}} \\
&= C2_{\mathrm{cur}}C1_{\mathrm{cur}}C0_{\mathrm{cur}}(R2_{\mathrm{cur}} \oplus C3_{\mathrm{cur}}),
\end{aligned}
\quad (7)
$$

$$I_{\mathrm{prv}} = I_{\mathrm{cur}} - 1. \quad (8)$$

For constraint length 9,

$$
\begin{aligned}
R2_{\mathrm{prv}}R1_{\mathrm{prv}}R0_{\mathrm{prv}} &= (R1_{\mathrm{cur}} \oplus C3_{\mathrm{cur}})VC4_{\mathrm{cur}}, \\
C4_{\mathrm{prv}}C3_{\mathrm{prv}}&C2_{\mathrm{prv}}C1_{\mathrm{prv}}C0_{\mathrm{prv}} \\
&= C3_{\mathrm{cur}}C2_{\mathrm{cur}}C1_{\mathrm{cur}}C0_{\mathrm{cur}}(R2_{\mathrm{cur}} \oplus C4_{\mathrm{cur}}), \\
I_{\mathrm{prv}} &= I_{\mathrm{cur}} - 1.
\end{aligned}
\quad (9)
$$

Constraint length 8

Constraint length 9

Constraint length 10

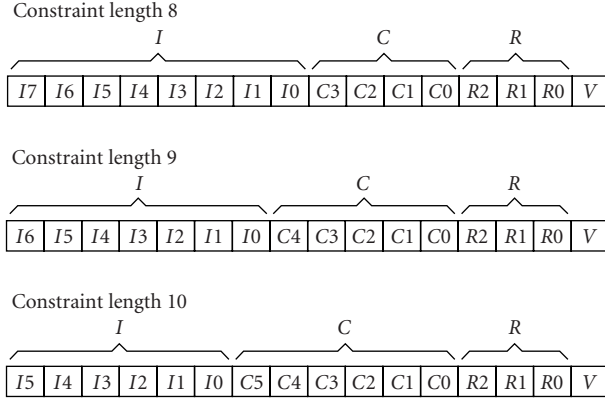
Figure 4: Traceback packets for constraint lengths 8, 9, and 10.

For constraint length 10,

$$R2_{prv}R1_{prv}R0_{prv} = (R1_{cur} \oplus C4_{cur})VC5_{cur}, \tag{10}$$

$$C5_{prv}C4_{prv}C3_{prv}C2_{prv}C1_{prv}C0_{prv}$$
$$= C4_{cur}C3_{cur}C2_{cur}C1_{cur}C0_{cur}(R2_{cur} \oplus C5_{cur}), \tag{11}$$

$$I_{prv} = I_{cur} - 1, \tag{12}$$

where the subscripts prv and cur denote the previous and current traceback steps.

From (2) to (12), we can see that, for each different constraint length, only two exclusive ORs and a down counter are needed to implement traceback mechanism. Moreover, two exclusive ORs can be shared by all constraint lengths for our configurable traceback SP module. In other words, the traceback logics of the configurable traceback SP module can be implemented by using four down counters (9-bit, 8-bit, 7-bit, and 6-bit), two exclusive ORs, and some multiplexers.

4.   IMPLEMENTATION RESULTS OF THE FPGA PROTOTYPE

In order to validate the configurable Viterbi decoder and evaluate its decoding performance, in terms of decoding delay, speed and resource usage, by using VHDL language, a synthesisable core of the decoder has been developed and implemented on Xilinx Virtex FPGA device [15].

The core's top-level interfacing is shown in Figure 5, in which the constraint length and the traceback depth can be instantly reconfigured through two configuration signals, ConstraintLength and TracebackDepth. *SDI1*[] and *SDI0*[] are data-input signals, each of which is 3-bit wide and corresponds to the received channel symbols (3-bit soft-decision quantisation is used). *Reset*, *Enable*, and *Clock* are global asynchronous reset signal, decoder core enable, and global clock signal, respectively. *BitOut* and *ValidOut* are decoded output signal and output status signal. Except *Reset*, all signals are synchronous to *Clock*, which is under the control of *Enable*. *Reset*, *Enable*, and ValidOut Signals are
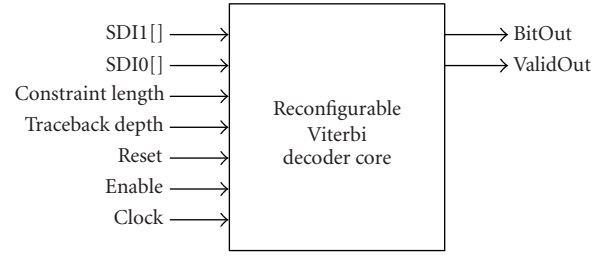

Figure 5: Reconfigurable Viterbi decoder core.

Table 11: The main specifications of our FPGA implementation.

Code rate (k/n)		1/2
Constraint length (K)		Configurable (7, 8, 9, and 10)
Traceback depth		Configurable (up to 62[a])
Soft-decision word length		3-bit
FPGA device		XCV300-6-PQ240
Frame size (bits)		Any size
Resource usage	slices	(1,137/3,072) 37%
	block memory	8
Maximum decoding frequency (MHz)		101

[a]The maximum traceback depths are 49, 57, 61, and 62 for constraint lengths 7, 8, 9, and 10, respectively.

active high. The decoding procedure is described as follows. Firstly, *Reset* must be applied to reset all internal states of the decoder before decoding and disable signal *ValidOut* by forcing it low. Secondly, with valid *Enable* signal, two 3-bit soft-decision channel symbols are latched into the decoder core via *SDI1*[] and *SDI0*[] at the rising edge of *Clock*, cycle by cycle. Finally, after an initial delay, the *ValidOut* signal becomes valid and the first decoded bit can be clocked out at the rising edge of the first clock with valid Valid-Out signal. Therefore, *Reset*, *ValidOut*, *Clock*, and *BitOut* can be used to implement a very simple external circuit to receive the decoded bits, which can be an output buffer if needed. *Reset* resets the external circuit to initial state. Whenever *ValidOut* is high, the decoded bits from *BitOut* can be latched into the external circuit at the rising edge of *Clock*.

In the FPGA prototype, the path metric RAMs are mapped onto Virtex distributed memory, while Virtex built-in block dual-port RAMs are used for survivor memory. One port is used to receive the survivor data from the ACS module and the other accommodates the traceback operation. This leads to a very simple and regular traceback architecture. The main specifications of the FPGA implementation are given in Table 11.

The decoding throughput and initial delay is given in Table 12. Obviously, it is the best possible decoding through-put rate for the area-efficient architecture with 8 ACS in

TABLE 12: Throughput rate and initial delay.

Constraint length	Throughput rate (bit/cycle)	Initial delay[a] (cycles)
7	1/8	507
8	1/16	770
9	1/32	1,677
10	1/64	3,380

[a]Initial delays are obtained from traceback depth of five times constraint length.

TABLE 13: Throughput rate and Equivalent gate count.

Viterbi decoder	Equivalent gates	Throughput (Mbps)
State-parallel fixed $K = 7$	87 836	32
Area-efficient fixed $K = 7$	26 208	13.5
State-parallel ($K = 3$–7) [8]	89 407	19.7
Area-efficient fixed $K = 10$	170 943	1.594
Area-efficient ($K = 7$–10)	172 618	12.625–1.578

parallel because no ACS is idle at any time. In addition, the proposed configurable Viterbi decoder can work with any size of frame data, so the initial delay could be ignored with a large enough frame.

To do BER testing, a PC-controlled BER testbench, as shown in Figure 6, has been developed which works in conjunction with the FPGA prototype. In order for the hardware testbench to be general and flexible, most functional modules such as message generation, FEC encoding, and channel model are implemented in software. Ethernet communication is used to download channel data to the hardware FPGA FEC decoder and upload the decoded results for decoding performance evaluation. BER results for constraint lengths with the traceback depth of five times the constraint length have been obtained and are shown in Figure 7. The measured BER results agree with the expected theoretical results [9].

5. COMPARISONS

Comparisons in terms of area (gates) and speed (throughput in Mbps) have been obtained from actual FPGA implementations. These are shown in Table 13. A fixed constraint-length ($K = 7$) Viterbi decoder was implemented using both a state-parallel and an area-efficient architecture with 5 levels pipelining using 8 ACS units to evaluate the pipeline scheme. With only 30% of the hardware resources of a state-parallel implementation, the area-efficient implementation achieved a throughput of 13.5 Mbps which is not too far off the theoretical expected rate ($5/8 * 32 = 20$ Mbps), taking into account the nonuniform delays across the FPGA. In order to evaluate the reconfiguration overhead, a fixed constraint length ($K = 10$) decoder was also implemented and comparisons were made with the reconfigurable decoder ($K = 7$–10). As shown in Table 13, the configuration overhead is only 1% while the throughputs are comparable.

The only previous work that is directly comparable to our work is the one reported in [8] based on a state-parallel implementation for constraints 3 to 7 only. From Table 13, for constraint 7, the throughput rate obtained in our case is inline with the expected ratio of 5/8 compared to the state-parallel implementation in [8]; of course a significant area overhead would be incurred by a state-parallel implementation for constraint lengths from 8 to 10.

Overall, the results obtained confirmed the design-space analysis in Section 2, taking into account that the prototypes are based on FPGA implementations. ASIC implementations would yield much more improved overall performance.

6. CONCLUSIONS

Broadband access raises new demands for channel coding. Besides higher decoding speed and decoding capability, reconfigurable decoding performance is highly desired, which suggests that decoding speed can be traded for decoding capability to adapt to the dynamic condition of a channel. In this paper, a novel design and implementation of an online reconfigurable Viterbi decoder has been proposed based on an area-efficient ACS architecture in which the constraint length and traceback depth can be dynamically reconfigured. A design-space exploration to trade off decoding capability, area, and decoding speed has been performed, from which the maximum level of pipelining against the number of ACS units to be used has been determined while maintaining an in-place path metric updating. A challenging example design with constraint lengths from 7 to 10 has been presented together with the new ACS schedule scheme, which provides 5 level ACS pipelining in this case and which can be applied for any constraint length in a totally uniform way. In general, this pipeline scheme can be applied to any area-efficient architecture with more than 8 time units for each ACS iteration. A modified variable shift path metric normalization and saturation protection are included in the ACS pipelining which allows for the path metric memory to be further reduced by 33% through using lower resolution for the path metric, compared with the case of modulo path metric normalization. In addition, best-state traceback is used to allow significant reduction of survivor memory. The design has been successfully implemented on Xilinx Virtex FPGA devices. FPGA implementation results, in terms of decoding speed, resource usage, and BER, have been obtained using a tailored testbench. These confirmed the functionality and the expected higher speeds and lower resources. Furthermore, the reconfigurable decoding performance, trading decoding speed, and area for decoding capability, has been verified. Further analysis will be carried out to confirm the expected improvement in power consumption offered by the proposed architecture.
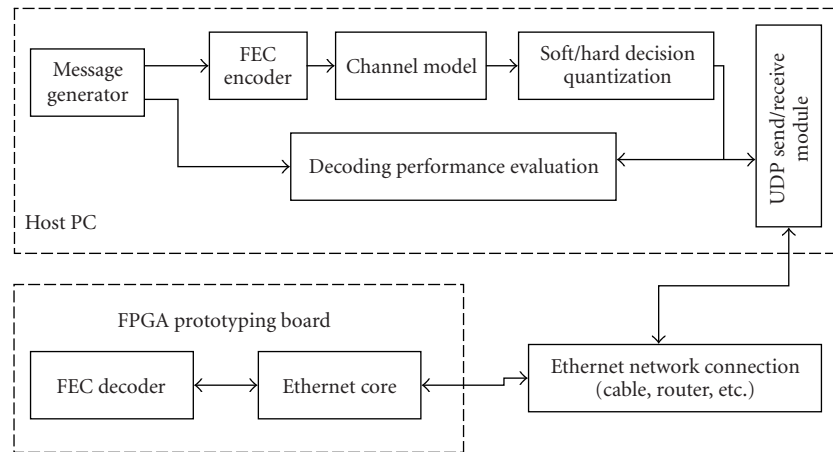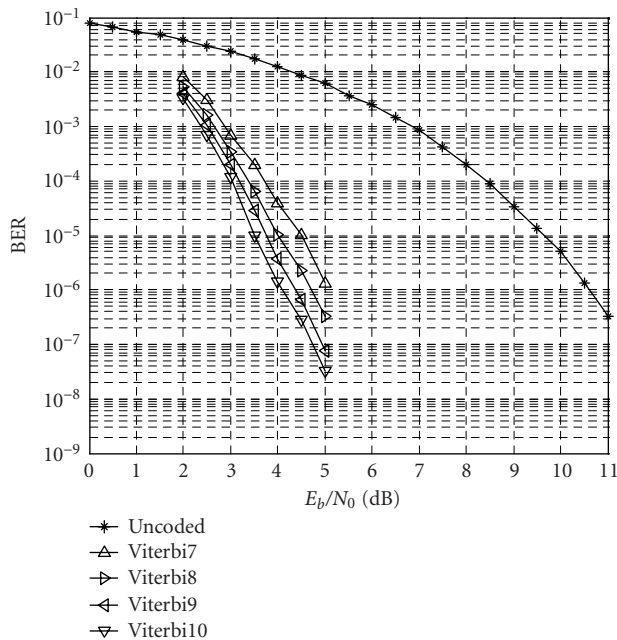
FIGURE 6: The block diagram of hardware testbench.

FIGURE 7: BER results of the configurable Viterbi decoder based on traceback depth of five times constraint length.

REFERENCES

[1] G. D. Forney Jr., "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[2] C. B. Shung, H.-D. Lin, R. Cypher, P. H. Siegel, and H. K. Thapar, "Area-efficient architectures for the Viterbi algorithm. II. Applications," *IEEE Trans. Communications*, vol. 41, no. 5, pp. 802–807, 1993.

[3] M. Bóo, F. Argüello, J. D. Bruguera, R. Doallo, and E. L. Zapata, "High-performance VLSI architecture for the Viterbi algorithm," *IEEE Trans. Communications*, vol. 45, no. 2, pp. 168–176, 1997.

[4] K. J. Page and P. M. Chau, "Folding large regular computational graphs onto smaller processor arrays," in *Advanced Signal Processing Algorithms, Architectures, and Implementations VI*, vol. 2846 of *Proceedings of SPIE*, pp. 383–394, Denver, Colo, USA, August 1996.

[5] P. H. Kelly and P. M. Chau, "A flexible constraint length, foldable Viterbi decoder," in *Proc. IEEE Global Telecommunications Conference*, vol. 1, pp. 631–635, Houston, Tex, USA, November 1993.

[6] M. Biver, H. Kaeslin, and C. Tommasini, "In-place updating of path metrics in Viterbi decoders," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 4, pp. 1158–1160, 1989.

[7] J. F. Arrigo, K. J. Page, Y. Wang, and P. M. Chau, "Adaptive FEC on a reconfigurable processor for wireless multimedia communications," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, pp. 417–420, Monterey, Calif, USA, May 1998.

[8] K. Chadha and J. R. Cavallaro, "A reconfigurable Viterbi decoder architecture," in *Proc. 35th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 66–71, Pacific Grove, Calif, USA, November 2001.

[9] G. C. Clark Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum press, NY, USA, 1981.

[10] S.-Y. Kim, H. Kim, and I.-C. Park, "Path metric memory management for minimising interconnections in Viterbi decoders," *Electronics Letters*, vol. 37, no. 14, pp. 925–926, 2001.

[11] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Communications*, vol. 37, no. 11, pp. 1220–1222, 1989.

[12] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. IEEE International Conference on Communications*, vol. 4, pp. 1723–1728, Atlanta, Ga, USA, April 1990.

[13] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, Radix-4 Viterbi decoder," *IEEE Journal of solid-state circuits*, vol. 27, no. 12, pp. 1877–1885, 1992.

[14] I. M. Onyszchuk, "Truncation length for Viterbi decoding," *IEEE Trans. Communications*, vol. 39, no. 7, pp. 1023–1026, 1991.

[15] Xilinx Corp., "Virtex 2.5V Field Programmable Gate Arrays Product Specification," http://www.xilinx.com.

Mohammed Benaissa is currently a Senior Lecturer in the Electronic and Electrical Engineering Department at the University of Sheffield. He is a member of the Electronic Systems Group. He has been actively working in the area of VLSI signal processing coding and cryptography for the past 15 years. He has published more than 40 papers in recognized journals and conferences. His recent research concentrate on investigating configurable approaches to optimum hardware implementation of error control coding and cryptographic techniques and their incorporation in SOCs.

Yiqun Zhu received the B.S. degree in electrical engineering and M.S. degree in image processing from Beijing University of Aeronautics and Astronautics, China, in 1988 and 1991, respectively. From 1991 to 1998, he worked in China Aerospace Corporation as a DSP Engineer. He is currently with the Electronic Systems Group, Department of Electronic and Electrical Engineering, the University of Sheffield, pursuing his Ph.D degree.

Special Issue on

Spectrum Sharing and Sensing for Future Broadband Networks: The Cognitive Radio Technology

Call for Papers

A recent measurement campaign on spectrum utilization promoted by the Federal Communication Commission throughout the US has revealed that significant portions of the electromagnetic spectrum are rarely exploited for considerable intervals of time, thus paving the way to a number of possibilities for increasing bandwidth utilization among multiple users in shared radio networks. The clue to improve spectrum utilization, as well as efficiency, comes from communications systems exploiting the so-called Cognitive Radio technology, which at the physical layer, as well as at upper-levels of the network, can adaptively and dynamically allow users to access radio resources by switching among portions of unused bandwidth in different intervals of time.

The cognitive radio technology is still in its infancy, and many problems at a theoretical, as well as practical, level have to be solved before this technology may be fully exploited in next generation wireless networks.

The main aim of this special issue is to highlight state-of-the-art techniques on the most recent research advances in cognitive radio networks.

Topics of interests include, but are not limited to, the following subject categories:

- Advanced algorithms on coding and modulation for software radio and cognitive radio networks
- Cross-layer algorithms based on spectrum sensing techniques
- Distributed algorithms for spectrum detection
- Information theory and performance limits of dynamic spectrum access
- Multimedia communications through cognitive networks
- Physical-layer design of software radio and cognitive radio transceivers
- Resource allocation for spectrum sharing in cognitive networks
- Spectrum management in cognitive radio networks

- Reconfigurable low-complexity transceiver architectures for channel adaptation in cognitive radio networks
- Radio test beds
- Regulatory policies on spectrum sharing for future broadband networks

Before submission authors should carefully read over the journal's Author Guidelines, which are located at http://www.hindawi.com/journals/ijdmb/guidelines.html. Prospective authors should submit an electronic copy of their complete manuscript through the journal Manuscript Tracking System at http://mts.hindawi.com/ according to the following timetable:

Manuscript Due	November 1, 2009
First Round of Reviews	February 1, 2010
Publication Date	May 1, 2010

Lead Guest Editor

Massimiliano Laddomada, Texas A&M University-Texarkana, TX 75505, USA; mladdomada@tamut.edu

Guest Editors

Hsiao Hwa Chen, National Cheng Kung University, Tainan City 701, Taiwan; hshwchen@mail.ncku.edu.tw

Fred Daneshgaran, California State University, Los Angeles, CA 90032, USA; fdanesh@calstatela.edu

Marina Mondin, Politecnico di Torino, 10129 Torino, Italy; mondin@polito.it

Hamid Sadjadpour, University of California, Santa Cruz, CA 95064, USA; hamid@soe.ucsc.edu

Special Issue on
Vehicular Ad Hoc Networks

Call for Papers

Recently, due to their inherent potential to enhance safety and efficiency measures in transportation networks, vehicular ad hoc networks (VANETs) have gained eye-catching attention from the wireless community. Traffic congestion wastes 40% of travel time on average, unnecessarily consumes about 2.3 billion gallons of fuel per year, and adversely impacts the environment. More importantly, traffic accidents are held responsible for a good portion of death causes. Annually more than 40 000 people are killed and much more injured in highway traffic accidents in the United States alone. Recently, intelligent transportation systems (ITS) have been proposed to improve safety and efficiency in transportation networks. The allocation of 75 MHz in the 5.9 GHz band for dedicated short-range communications (DSRC) by the FCC was a move toward this goal, which was further complemented by the introduction of the vehicle infrastructure integration (VII) initiative by the US Department of Transportation. VII proposes to use dedicated short-range communications (DSRC) to establish vehicle-to-vehicle and vehicle-roadside communications to deliver timely information to save lives, reduce congestion, and improve quality of life.

Despite the much attracted attention, there still remains much to be done in the realm of vehicular ad hoc networks. Signal processing plays a major role in vehicular ad hoc networks. The aim of this special issue is to present a collection of high-quality research papers in order to exhibit advances in theoretical studies, algorithms, and protocol design, as well as platforms and prototypes which use advanced signal processing techniques for vehicular ad hoc networks. Topics of interest include but are not limited to:

- Estimation and detection techniques in VANETs
- Localization techniques in VANETs
- Clock synchronization in VANETs
- Security and privacy in VANETs
- Sensing in vehicular environments
- Channel modeling for V2V communications
- MAC, routing, QOS protocols, and analysis for VANETs
- VANET smart antenna technologies
- Dynamic spectrum access and cognitive radios for VANETs
- Congestion control and cooperative VANETs
- Traffic modeling in VANETs
- Signal processing to utilize data correlation in VANETs
- High-speed (rapid) signal processing for VANETs
- Accurate/high-fidelity simulation of VANETs
- Signal processing considerations in real world deployments of VANETs

Before submission authors should carefully read over the journal's Author Guidelines, which are located at http://www.hindawi.com/journals/asp/guidelines.html. Prospective authors should submit an electronic copy of their complete manuscript through the journal Manuscript Tracking System at http://mts.hindawi.com/ according to the following timetable:

Manuscript Due	November 1, 2009
First Round of Reviews	February 1, 2010
Publication Date	May 1, 2010

Lead Guest Editor

Hossein Pishro-Nik, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, 100 Natural Resources Road, Amherst, MA 01002, USA; pishro@ecs.umass.edu

Guest Editors

Shahrokh Valaee, Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, ON, Canada M5S 3G4; valaee@comm.utoronto.ca

Maziar Nekovee, Complexity Group, BT research, Polaris 134 Adastral Park, Martlesham, Suffolk IP5 3RE, UK; maziar.nekovee@bt.com

Special Issue on
High-Throughput Wireless Baseband Processing

Call for Papers

Wireless communications is a fast-paced area, where many standards, protocols, and services are introduced each year. Implementation of every new standard becomes challenging especially when more and more higher data rates up to several gigabits/second are required. On the other hand, the power budget is not increasing in the same pace. The presence of all those different modes as well as high throughput requirements brought the need for designing almost-all-digital radios, which benefit from technology scaling. Those goals can only be achieved by efficient algorithms, models, and methods for the design of high-throughput and low-power systems for baseband processing. This special issue will report the recent advances of very high throughput and low-power systems for wireless baseband processing. Areas of interest include, but are not limited to:

- Modeling of quality-of-service, reliability, and performance in high-throughput wireless systems
- Power-aware and/or low-cost algorithms and architecture optimizations for multistandard baseband processing
- Baseband compensation techniques for RF/analog circuit impairments
- High-throughput baseband processing for software-defined and cognitive radios
- Applications to WirelessHD, IEEE 802.15.3c, MIMO systems, UWB, WiMAX, and LTE systems

Before submission authors should carefully read over the journal's Author Guidelines, which are located at http://www.hindawi.com/journals/wcn/guidelines.html. Prospective authors should submit an electronic copy of their complete manuscript through the journal Manuscript Tracking System at http://mts.hindawi.com/ according to the following timetable:

Manuscript Due	October 1, 2009
First Round of Reviews	January 1, 2010
Publication Date	April 1, 2010

Lead Guest Editor

Taskin Kocak, University of Bristol, Bristol, UK; t.kocak@bristol.ac.uk

Guest Editors

Mustafa Badaroglu, ON Semiconductor Vilvoorde, Belgium; mustafa.badaroglu@onsemi.com

Dake Liu, Linkoping University, Linkoping, Sweden; dake@isy.liu.se

Liesbet Van der Perre, IMEC, Leuven, Belgium; vdperre@imec.be