A New Approach of a Self-Timed Bit-Serial Synchronous Pipeline Architecture

Achim Rettberg, Mauro Zanella, Christophe Bobda, Thomas Lehmann

University of Paderborn, Paderborn / Germany
Email: achim@c-lab.de, zanella@mlap.de, bobda@upb.de, Thomas.Lehmann@torkin.de

Abstract

Power consumption, area minimization as well as signal delay and reconfiguration with respect to rapid system prototyping make increasing demands on chip design. While design space can be reduced by bit-serial operators, long control lines in synchronous bit-serial architecture usually affect the performance of the circuit. This paper presents a new synchronous, fully reconfigurable self-timed bit-serial and fully interlocked pipeline architecture. Through a one-hot implementation of the central control engine, we realize the local control of the operators. Furthermore we developed specialized routing components that allows the reconfiguration of the implementation w.r.t. to rapid system prototyping. This realization of the developed architectures provides the freedom of a rapid system prototyping of a given problem. To our knowledge, this is the second paper detailing the implementation of a fully interlocked synchronous architecture after the one by Jacobson et al. [1] and the first which does not rely on gated clocks to realize the local control of the operators. We prove the usefulness of our architecture by an example implementation of a given problem on a Xilinx Virtex 400E FPGA. The architecture is tailored to the control in mechatronic systems. High speed, parallelism, re-programmability and compactness are the main characteristics of this architecture and vital for the realization of embedded systems for control purposes. As an application example this paper presents the implementation of a PI controller on a Xilinx Virtex 400E FPGA. This architecture can of course also be used in any other field of application.

This contribution is organized as follows: Section 2 describes the architecture. An example of a PI controller is given in section 3. The final section comprises a summary and an outlook.

2 Concept of the Architecture

2.1 General Overview

The combination of different design paradigms and the reassignment of design methods leads to new architectural concepts. In our approach we combine the synchronous and the bit-serial design paradigm. Furthermore, we adapted to our approach and concepts of operators and configuration elements to advance asynchronous bit-serial architectures.

One field of application of such an architecture is e.g. signal processing in terms of digital filters or digital controllers. These algorithms may be realized in hardware by means of the proposed architecture; this requires only small chip area and an equally small number of input and outputs pins, thus reducing the size and the complexity of the printed circuit. The speed of the bit-serial processing is high enough for the application domain in question. E.g., in an electrical motor-current control there are the delays of input/output converters (e.g., A/DS and D/As) and those resulting from the inertia of the motor.
2.2 Self-Timed Bit-Serial

As a rule, mapping huge dataflow graphs to the target architecture requires a central control unit. This unit provides synchronization between the operators and controls data distribution and storage within the realized circuit. Usually, the control unit is realized by a finite-state machine. This kind of control-unit implementation brings about long control wires with signal delays and high complexity in the design of such a unit. In particular signal delays are a crucial problem of today’s chip implementation, because signal delays due to the length of the wires are in the range of gate delays. In combination with an increase in chip area it may not be possible to implement a central control unit. Therefore, the aim is to realize all the control signals locally.

In the architecture presented, the central control unit is replaced by local control elements. These are synchronized and interlinked by handshake wires. Therefore the required control wires are local. As mentioned before, the handshake mechanism used in the architecture is similar to the one that is used in bit-serial asynchronous architectures (cf. [4], [5] and [6]).

The central control unit may be conceived as a simplified counter that, at a given time, triggers defined actions in given time slots. This reflects the paradigm of synchronous design. Taking into account a so-called one-hot implementation, it is possible to map the counter to a defined shift register with a special marker that is “pushed” through the register. When the marker reaches a defined position within the shift register it initiates actions in the circuit. The main idea of this architecture is to map the counter (control unit) to a shift register in the data path. Bit-serial shift registers are necessary for a bit-serial implementation of the data path. The modified operators in the data path recognize the marker of the counter. Therefore, the marker can pass through the operations unhindered and unmodified. It is attached to the data and controls the data processing in the data path. A data package contains the control marker and the processed data. Additionally, a gap may be included between the control marker and the data. Such a gap can be regarded as a separator. A valid data package is shown in Figure 1.

![Figure 1. Assembly of a data package with a control marker, a gap, and data](image1)

The control marker is shifted through the data path together with the data and the gap. The bit order of the data requires LSB\(^1\) first. To recognize the control marker in the data path, so-called scanners are needed (see Figure 2). These scanners identify the marker and activate the component control. The time factor is mapped to a position within the implemented data path; thus we know exactly the length of the control marker, the gap and the data and can therefore detect where the data are processed when a marker signal reaches a certain position in the data path. Hence, the proposed architecture works in bit-serial mode. The distance between the control information (the marker) and the operations is defined with the operations to be affected only by the control marker of the actually processed data. Consequently, the distance between control wires and the controlled elements is locally predefined.

![Figure 2. Scanner implementation](image2)

We have to make sure a control pattern is well-defined so as to avoid problems in the case there are two or more identical patterns within the control marker in the circuit. To solve this problem we use two scanners (see Figure 3) that realize an insensitive automaton against those patterns. The first scanner activates the automaton and the second ends the activation phase. The automaton goes into idling state (wait). An edge detection allows retaining the original scanner signal (see Figure 4). Both edges can be used as a control signal depending on the required functionality.

![Figure 3. Two scanners with the corresponding automaton and the detection of a positive edge](image3)

\(^1\) LSB = least significant bit
It is necessary that the scanners observe a minimum distance between each other because a reset to the "wait" state of the automaton can only be done if no other control marker is identified in the data package; this activates the automaton (see Figure 3). Furthermore, the data packages have also a minimum distance between each other because a signal pattern from a preceding data package must not reset the automaton. All control signals of an operation with valid data are activated by a control marker of the corresponding data package.

![Figure 4. Signal characteristics of the scanners, the automaton, and the edge detector](image)

Data packages moving relatively independently along the data path have to be synchronized by non-unary operations in view of compensating for the different path lengths. For this purpose, the fastest path in the dataflow graph is blocked by a so-called stall wire until all other necessary data packages have arrived at a specific synchronization point. This functionality is realized by a component named synchronizer (see section 2.3). Due to the defined length of the data packages the length of the stall wire is locally limited.

The locally limited impact of control information and the limited blocking of operations in the architecture allow the implementation of a pipeline architecture, i.e., several data packages can be shifted one after the other in a sector of the dataflow graph. The only constraint is that all data packages have a minimum distance between each other, thus preventing interaction.

In the following sections we describe how we implemented the synchronizer element, the operations and the shift register that are used within the circuit and for the communication with the environment.

2.3 Synchronizer Implementation

As discussed in the previous section it is mandatory to synchronize the inputs into non-unary operations in bit-serial architectures because the data have to be shifted in parallel through the operators. This synchronization mechanism is realized by a synchronizer component (see Figure 5). The function of the component is to block the inputs that arrived earlier until all inputs are ready. The stall wire in Figure 5 realizes the blocking of the corresponding inputs. When all inputs have valid data and are ready, the stall signal is altered and the data are synchronized and shifted to the operator by the synchronizer component. The valid data at the inputs are recognized by the control marker that is stored before the gap and the data in a data package. Here, the control marker can be interpreted as a synchronization marker.

![Figure 5. Synchronizer realization](image)

![Figure 6. Synchronizer communication (data packages can be released if the automaton in the middle is in the state y)](image)

A controller of the synchronizer assumes one of two states: one for waiting for all data packages (wait) and the
other for pushing them (push). Figure 5 shows the internal implementation of the synchronizer. A synchronizer has a control unit and a scanner for each input. A scanner recognizes if a control marker arrives at the input. When all scanners in the synchronizer detect a control marker, the control unit switches to the stage push. Otherwise, the inputs with valid data are blocked and the control unit switches to the state wait. In this case, the synchronizer sends zeroes to the subsequent operator. This ensures that the circuit behind the synchronizer is not blocked. Another scanner in the data path resets the signal to wait. This ensures that the data packages arrive at the operator synchronously.

It is necessary to have a minimal difference between the scanner(s) on the input side and the one(s) that resets the signal. The data package has to be stored within this range. Furthermore, we have introduced a block-stall signal in the architecture. "Block-stall" means that a complete block with scanners on the input and output sides of an operator is blocked. The counterpart of the block-stall is the block-free signal. In the case of the latter, the block can be used. Additionally, realization of the synchronizer requires the inputs to be synchronous and the block to be free (block-free is true). In this case the synchronizer reads the data packages and sets the block-stall signal. This leads to a communication and evaluation of scanner signals between the different synchronizers (see Figure 6).

2.4 Router Implementation

In order to realize reconfiguration within our architecture we developed a component called router. The router component offers the freedom to select different paths within the implemented design. With the help of the router it is possible to map different algorithms represented each by a dataflow graphs into a single dataflow graph.

Figure 7. Extended control marker

The selection of the paths within the design can be controlled by the extension of the control marker in the data packet. That mean, the control marker contains the routing information, see Figure 7. It is necessary to bypass the control marker at each operator, see Figure 8. Otherwise the control marker will be modified. This bypassing is controlled by the bypass trap signal, that is generated by the start-bit of the control marker at a specific point in the dataflow graph. In Figure 8 the control marker is bypassed at the addition operator (+). The stall signal goes directly through the router, see section 2.3. At the joint element in Figure 8 we have to ensure that data packets with the same control marker arrived. Only at the router component the routing information of the control marker can be modified.

Figure 8. The router component and a join element

The router component is configurable by the output port names. For example, the router 1 in Figure 9 has the output ports u and v. That mean, in the control marker has to be stored if the data packet is routed to output port u or v.

Figure 9. Routing example in a dataflow graph

Now we decide between to different ways to save the routing information in the control marker. In the first
approach all output ports of the routers for the path to be realized in the dataflow graph is stored in the control marker. For example, if we want to route in the dataflow graph of Figure 9 over port \( v \) of router 1 and port \( y \) of router 3 the control bit-vector \( \langle v, y \rangle \). Whereat the control marker is not modified. In the second approach each router consumes his own part of the control bit-vector and deletes his entry in the marker. In our example that mean, router 1 deletes \( <v> \) in the control-bit vector.

2.5 Operator Implementation

With operator implementation, cost-effectiveness and performance are vital factors. Each operator has a synchronizer component at the inputs (see Figure 5).

There are tradeoffs between parallel and serial operators. As parallelism increases performance and implementation size, an interesting approach is to combine a bit-serial operator implementation with a pipelining which reintroduces parallelism. A bit-serial operator is small and can be used for different data-widths. In this section we describe briefly a bit-serial, constant coefficient multiplication component which is implemented as a repeated addition (Figure 10).

This implementation of the multiplication is similar to the asynchronous multipliers described in [7]. Each type of bit-serially implemented multiplication has to be adapted linearly to the data-width; this kind of implementation reduces the complexity from \( O(n^2) \) - for a parallel multiplier - to \( O(n) \). Thus the complexity of the multiplication operators increases linearly because exactly one addition block per bit of the data word is instantiated (see Figure 10). Three slightly different addition slices have been developed. The first slice type duplicates the input data stream. The second type reads two operands, stemming from the original data stream, and the result of the preceding slice. The multiplication is finished by the last addition and only the result is passed on. All three slice types can be configured by means of a configuration cell (CMC) which consists of standard flip-flop registers; when queued, these make up a shift register. The stored value determines if the original data have to be added to this slice. In principle, the addition slices realizing the multiplication step for a configuration value of “0” in the CMC can be implemented more efficiently but then the option of configuration will be lost. Our implementation, however, is simpler than a general multiplier because matching the constant coefficient bits with the input data bits is trivial.

Figure 10. Bit-serial constant coefficient multiplier (only data path is displayed)
3 Realized Example: PI controller

Mechatronics, being a multidisciplinary field of engineering, performs the complex task of combining the classical domains of mechanical engineering, electrical engineering, control engineering, and computer science.

In order to obtain manageability of the complex structure of a mechatronic system, one approach to the mechatronic design is to decompose the entire system into subsystems with regard to their different functions. The concept of function in mechatronics can in most cases be seen as a synonym for the desired controlled motion behaviour of a system. A proposal for the structuring of mechatronic systems as a basis for a generalized design method is presented in [2]. The approach discussed there is a combination of hierarchical and modular organization, with a distinction being made between the mechatronic system components.

On the lowest hierarchical level, physically related sensors, actuators, mechanical structures, and information-processing units are combined to make up Mechatronic Function Modules (MFM). The information processing of the control has to work under hard real-time conditions. One example of an MFM can be the local control of a linear DC motor.

For test purposes we have realized a PI (Proportional and Integral) controller [3]. The transfer function of the PI compensator, \( G_c(s) \), is given by:

\[
g_c(s) = K_p + \frac{K_i}{s} \tag{1}
\]

Observe from Eq. (1) that the PI compensator provides a proportional term and an integration term, as its name implies.

Assuming that the input to the PI compensator for a given step \( t \) is \( u(t) \) and its output is \( y(t) \), the equation defining the operation of the PI compensator is given by:

\[
y_t = u_t K_p + u_{t-1} \frac{1}{s} K_i \tag{2}
\]

We have implemented the controller on an FPGA. For this purpose it was necessary to transform the continuous integrator \( s \) into a discrete-time integrator \( z \). Applying the trapezoidal integration method for a given sampling period \( T \), the PI compensator in a discrete form is given by:

\[
y_t = u_t K_p + K_i (y_{t-1} + \frac{T}{2} (u_t + u_{t-1})) \tag{3}
\]

The PI compensator realized on the FPGA by means of the architecture described in this paper is shown in Figure 11.

![Figure 11. Realized PI controller](image)

In mechatronic systems, the sampling rate of the control depends on the inertia of the mechanical system, as well as of the delay of sensors and actuators (converters). For this implementation the target platform is the FPGA module of the RABBIT system [8]. This module features a Xilinx Virtex 400E, 16-bit A/D converters with 100 kHz and 14-bit D/A converters with the same sampling rate. The PI controller occupies 7% of the total amount of FPGA slices. In view of the timing of the A/D and D/A converters, the maximum sampling rate of the PI control can even be 100 kHz.

The speed of the design is approximately 50 MHz. Taking into account the entire overhead related to the control bits of a data package, 16-bit data will be processed by this implementation in about 500 kHz. This may be slower than a parallel implementation, but much faster than required by such a system.

4 Conclusion and Outlook

This paper presented a novel implementation of a bit-serial processing architecture. This architecture has the peculiar feature of being self-timed and comprises a fully interlocked pipelining structure which aims at controlling the different computational paths of a system design. We have shown how useful this approach is in terms of chip area, low-power design, and speed.

Furthermore the architecture allows the mapping of different dataflow graph into one graph by including router in the graph and routing information in the data packet. This offers the freedom of reconfiguration within the implementation and supports also rapid system prototyping.

For the design of embedded systems, to be applied for control purposes in mechatronic systems, this new architecture represents a novel approach, especially as regards the distribution of controllers and data information. One example is the automotive industry where performance, space, cost, size, and weight are of vital importance. With the architecture described it is possible to implement
different controllers on different hierarchical levels (e.g., in cascade control) on one or more chips in a transparent way.

References


