TR-FSM: Transition-based Reconfigurable Finite State Machine

JOHANN GLASER, MARKUS DAMM, JAN HAASE and CHRISTOPH GRIMM
Institute of Computer Technology, Vienna University of Technology, Austria

Finite state machines (FSMs) are a key element of integrated circuits. Hard-coded FSMs do not allow changes after the ASIC production. While an embedded FPGA IP core provides flexibility, it is a complex circuit, requires difficult synthesis tools and is expensive. This paper presents and evaluates a novel architecture that is specifically optimized for implementing reconfigurable finite state machines: Transition-based Reconfigurable FSM (TR-FSM). The architecture shows a considerable reduction in area, delay and power consumption compared to FPGA architectures with a (nearly) FPGA-like reconfigurability.

Categories and Subject Descriptors: B.5.1 [**Register-Transfer-Level Implementation**]: Design—*Control design*; B.1.1 [**Control Structures and Microprogramming**]: Control Design Styles—*Writable control store*; B.6.1 [**Logic Circuits**]: Design Styles—*Sequential circuits*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Standard cells*

General Terms: Design, Performance, Theory

Additional Key Words and Phrases: Reconfigurable logic, Finite state machine, FPGA, Implementation, Low power, Wireless sensor network

1. INTRODUCTION

Digital designs often require finite state machines (FSMs). The hard-coded design is common practice but does not allow any changes after chip production. The exertion of programmable logic such as an embedded FPGA would permit a trade-off, but unfortunately introduces a large area and power overhead [Hartenstein 2001]. In this paper, a new architecture for implementing re-configurable FSMs is presented.

We consider FSMs with n_S state bits, n_I input signals, and n_O output signals, with the respective signals being Boolean. The state transition function and the output function derive the next state and the output signals from the current state and the input signals. The number of possible transitions per state equals 2^{n_I}. Since there are 2^{n_S} possible states, we get an upper bound of $n_T \leq 2^{n_S+n_I}$ for the total number of transitions for such an FSM. However, FSMs in concrete applications have a number of transitions which is considerably lower than this bound. This is especially true for FSMs with many input signals, where certain states are only

This work is conducted as part of the Sensor Network Optimization through Power Simulation (SNOPS) project which is funded by the Austrian government via FIT-IT (grant number 815069/13511) within the European ITEA2 project GEODES (grant number 07013).

inspecting a subset of these signals.

For FSMs with such low transition-per-state-ratios, we proposed a novel reconfigurable architecture which focuses on the *transitions* rather than on the state transition and output *function*, and call this *Transition-based Reconfigurable FSM (TR-FSM)* [Glaser et al. 2010]. Instead of providing a big reconfigurable block for implementing the whole state transition function, we provide several smaller reconfigurable blocks for implementing each transition.

The rest of the paper is organized as follows: The next section surveys hardware implementations for reconfigurable FSMs. Then the new architecture is described, followed by a summary of extensions. In Sec. 4 a hardware implementation featuring an assistance module for a wireless sensor network node is presented and evaluated. Finally a summary is given.

2. RELATED WORK

For the implementation of an FSM in an ASIC, synthesis tools use logic minimization algorithms to find optimal combinational circuits which realize the state transfer function and the output function. An alternative implementation is to read from a ROM to retrieve the results of the two functions. Both, the input signals and the state signals are used as address inputs. The data output of the ROM is split into the FSM output signals and the next state signals, where the latter are fed back to the inputs through a clocked register [Katz 1994].

A RAM in read mode is a simple approach to realize a *reconfigurable* FSM. By writing the RAM content to a specific set of data the FSM functions are specified. The disadvantage of the memory approach is the required size. For n_I input signals, n_S state bits and n_O output signals, the memory has to offer $2^{n_I+n_S}$ words with a width of $n_S + n_O$ bits.

The first reconfigurable logic structures were implemented as sum-of-product term structures in PALs and PLAs. Another widely used circuit design for reconfigurable logic are FPGAs. These comprise look-up tables, flip-flops and rich routing resources to universally implement any logic function [Rabaey et al. 2003].

According to Bukowiec [2008] the synthesis results for FSMs in an FPGA are suboptimal. Therefore, new synthesis methods using multi-level architectural decomposition and utilization of Block RAMs to reduce the amount of logic resources are introduced. While this thesis is an important step towards FSM optimization for reconfigurable architectures, it still relies on fine-grained FPGA structures which impose large configuration and routing overhead and a power and area penalty [Hartenstein 2001].

In Liu et al. [2005], an unbalanced and unsymmetrical architecture for reconfigurable FSMs is introduced. The FSM is split into a sequential section which implements the state transition function and a logic section which implements the output signals, both connected via routing resources. Although this approach achieves an area reduction of 43 % and a power consumption decrease of 82 %, it has overhead due to its concentration on the logic functions for the next state and output signals. The approach was further optimized for FSMs with a high number of states [Liu et al. 2006].

While the previously mentioned approaches implement the full FSM at once, Mil-

ligan and Vanderbauwhede [2007] only implement the logic required to calculate the next state (and output signals) for the current state. After every state transition, the internal logic is reconfigured to realize the next state logic for the current state. While this approach greatly reduces the total amount of logic elements by increasing their utilization, the permanent reconfiguration effort is not a low power approach.

3. TRANSITION-BASED RECONFIGURABLE FSM ARCHITECTURE

The basis of the proposed architecture is the so called *transition row* (see the dash-dotted boxes in Fig. 1). A state selection gate (SSG) compares the current state to its configured value and enables the transition row. A reconfigurable input switching matrix (ISM) selects a subset of n_T out of the n_I input signals which then serve as the inputs for the input pattern gate (IPG), which is a reconfigurable combinational logic block, e.g. a look-up table (LUT). If activated, the IPG outputs a "1" iff the input pattern matches a certain transition from the recognized state. We also refer to n_T as the *size* of the transition row.

The overall architecture contains many such transition rows with varying input width (see Fig. 1). The number of transition rows and their input widths are derived during the specification stage. All reconfigurable *next state ID registers* (NSRs) of the transition rows are fed to a multiplexer ("Select"). It selects the NSR output value of the transition row for which the IPG outputs a "1". This next state is fed back to the *state register* (SR) as the new state of the FSM. The according output pattern register (OPR) is also selected by this multiplexer and used as FSM output.

As depicted in Fig. 1, the current state is also fed into the next state selection logic block ("Select"). This allows for a very simple treatment of loops, i.e. transitions where the FSM remains in the same state. In the case that none of the transition rows are activated, the select logic chooses the current state as new state. Therefore, no transition rows have to be used for loops.

To summarize, the reconfigurable parts of the architecture are the state selection gate (SSG), the input switching matrix (ISM), the input pattern gate (IPG), the next state ID register (NSR) and the output pattern register (OPR).

Applying TR-FSM for ASIC or SoC-design is a two-stage process. In the first stage, the necessary resources for the TR-FSM are specified. This can either be an estimation based on FSM prototypes for a certain application class, or it is based on a collection of FSMs which the resulting TR-FSM must be able to be configured to. From this specification, the semiconductor chip containing the TR-FSM is produced. In the second stage, this TR-FSM embedded in the chip now can be configured with an actual FSM analogous to configuring an FPGA. However, since the TR-FSM is already structured like an FSM, the synthesis process is considerably less complex.

This architecture limits the total number of transitions in a specific FSM. Although it is possible to use each transition row for any starting state. Therefore we can trade off the number of states by the number of transitions per state. That is, with a fixed chip an FSM with many states and few transitions can be implemented as well as an FSM with few states and many transitions, and anything in between.

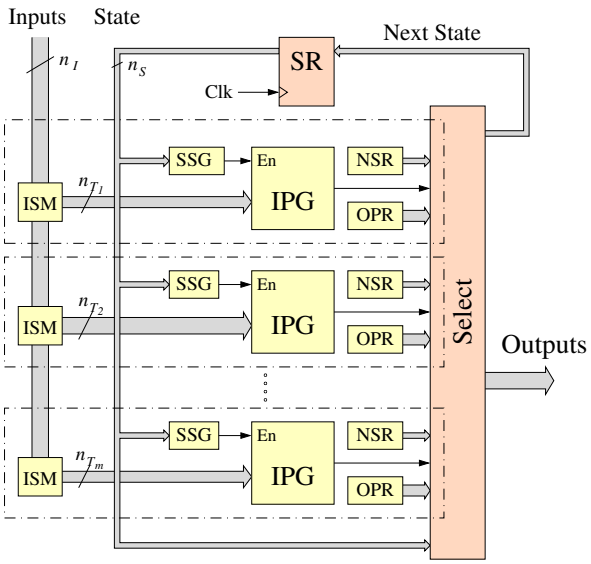


Fig. 1. The overall TR-FSM architecture

3.1 Implementation Details

3.1.1 *State Selection Gate (SSG).* Since the SSG has to recognize one specific state, it can be implemented with an n_S bit wide AND gate preceded by configurable input inverters. However, an FSM might have a common error state which is reached from several (but not all) states if an error input is active. For such multi-source transitions, an SSG implemented as an n_S input LUT instead of an AND gate can be used. Now, the respective transition row can be enabled for multiple states.

3.1.2 *Input Pattern Gate (IPG).* The IPG can be implemented in different ways. The first implementation is similar to that of the SSG, i.e., a n_T wide AND gate with configurable input inverters. This gate can recognize only one specific input pattern.

The second implementation is to use a LUT, such that it is possible to activate the transition for multiple different input patterns. This is beneficial either if the transitions from one state have mixed don't-care conditions or if an IPG with more inputs than necessary is used (e.g., because all rows with less inputs are used by other transitions).

3.1.3 *Input Switch Matrix (ISM).* Consider an ISM with n_I input signals of which $n_T < n_I$ are connected to the IPG, requiring n_I switches per connected signal. Since only one (or none) of the n_I input signals is selected, there are $n_I + 1$ possible switch combinations, which can be encoded as binary numbers with $\lceil \log_2(n_I + 1) \rceil$ bits. Altogether, this yields $n_T \cdot \lceil \log_2(n_I + 1) \rceil$ configuration bits per transition row. If we exclude the possibility to select no signal (e.g. if the IPG is realized as a LUT such that it can implement don't-cares), this number reduces to $n_T \cdot \lceil \log_2(n_I) \rceil$. We call this *binary encoding*.

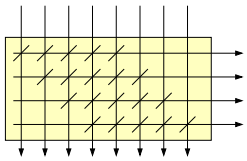


Fig. 2. Optimizing the Input Switch Matrix.

Another variant to encode the configuration of an ISM exploits that the order of the input signals for the IPG is not important. Therefore, a bit vector of size n_I of which a maximum of n_T bits are set to "1" is sufficient to configure the ISM. Every bit represents a primary input which is connected to the IPG if its respective bit is "1". However, this *boolean encoding* is only more concise than the binary encoding if $n_I < n_T \cdot \lceil \log_2(n_I + 1) \rceil$, i.e. for TR-FSMs with a large n_I and for transition rows of small size the binary encoding is more beneficial.

The number $n_I \cdot n_T$ switches per ISM can also be optimized due to the insignificance of the order of the selected inputs (see Fig. 2): If the leftmost input signal is selected, it can be selected as first IPG input signal, and since it can be selected only once, no further switches are required for the leftmost input signal. The same holds for the second input signal on the left and the second IPG input signal, and so on. Also, the rightmost input signal can always be handled by the last IPG input signal, therefore the respective switch can be removed from the previous IPG input signals, and so on. Altogether, from every selection signal, $n_T - 1$ switches can be removed, such that the total number of switches is $[n_I - (n_T - 1)]n_T$, which gives a reduction of $n_T(n_T - 1)$ switches.

From this reduction of switches for every input of an IPG the binary encoding can also benefit: The number of configuration bits is then given by $n_T \cdot \lceil \log_2(n_I - n_T + 1) \rceil$.

3.1.4 *Next State Selection.* Every transition row holds the configurable next state ID register (NSR). A multiplexer selects the next state ID associated with the transition row which outputs a logical "1". The configuration applied by the bit stream must ensure, that for every combination of inputs and states at most one of the IPG emits a logic "1" and all others emit a logic "0". The Output Pattern Register (OPR) is analogous to the NSR.

3.2 Analysis

We analyzed the proposed approach using several benchmark FSMs and compared it to FPGA synthesis results [Bukowiec 2008]. We chose FSMs which need at least 4 state bits and had relatively few transitions per state. These FSMs were grouped into 3 sets with roughly similar values regarding state vector width, input signals, and output signals. The TR-FSMs needed considerably less configuration bits of those needed by FPGAs (only 7.3 to 12.7 %) [Glaser et al. 2010]. While we see this as an indicator for a prospective improvement regarding area and power consumption, this is also beneficial by itself regarding run-time reconfigurability and for applications with limited memory resources like wireless sensor nodes.

4. HARDWARE IMPLEMENTATION

For the evaluation of the key concepts of our previous work [Glaser et al. 2010; 2010; 2010] a test chip was produced. This also includes TR-FSM instances which will be described in this section in further detail.

4.1 Application Example

For a Wireless Sensor Network (WSN) node the energy consumption is a major design criterion, since batteries are difficult to replace and the energy harvesting potential is low. Typically a WSN node comprises of a CPU, memory, RF transceiver, and some peripherals like a timer, analog-to-digital converter (ADC) and digital IO. The CPU executes the firmware and controls the whole node. Usually it is placed in an inactive low-power mode and is only activated for short periods. When activated sensor measurements and communication tasks are performed. These activities include several simple sub-tasks like setting digital output signals, waiting for transitions on digital input signals and comparing integer numbers. For all these simple tasks the CPU is certainly overqualified while the waiting periods even consume energy for doing nothing. Therefore we propose to equip a WSN node ASIC with dedicated logic blocks to accomplish these simple subtasks. These take over the periodic wake-ups and simple processing from the CPU and only activate the CPU if further, more complicated processing like creating a network packet, is required.

Unfortunately shifting the hardware-software-partitioning decision towards hardware reduces the flexibility, because a silicon chip can not be modified after production while the firmware can easily be replaced in flash memory. An ASIC could only be used for a single application with predefined external components (e.g. sensors). If a bug was found, an expensive chip redesign would be necessary. Therefore we propose to make the assisting modules reconfigurable to re-establish the required flexibility.

A WSN node can contain one or more reconfigurable assisting modules where each is specifically designed to fulfill a predefined application, e.g. sensor measurements or MAC protocol handling. Therefore a so called *application class* is defined for every module which specifies the field of planned tasks performed by the module. From this the hardware circuit is derived. After the ASIC is produced, the actual application is defined, and together with the reconfigurable resources of the ASIC the configuration is created. On startup of the chip the configuration is applied which activates the functionality of the reconfigurable module.

One typical application for a reconfigurable module is the sensor interface. In periodic intervals an external sensor is activated. Then the ADC is started to sample and convert the signal to a digital value. After waiting for the completion of the conversion, the new value is compared to the previous value. If the difference is below a certain threshold, the job is done. If the value has changed more than the threshold, this new value is stored and the CPU is notified. It will take over control and execute more complex tasks, like transmitting the new value via the wireless network. To investigate this example application with one silicon chip but varying details (e.g. using an analog sensor and an ADC or using a digital sensor with an I^2C interface), the test chip was produced.

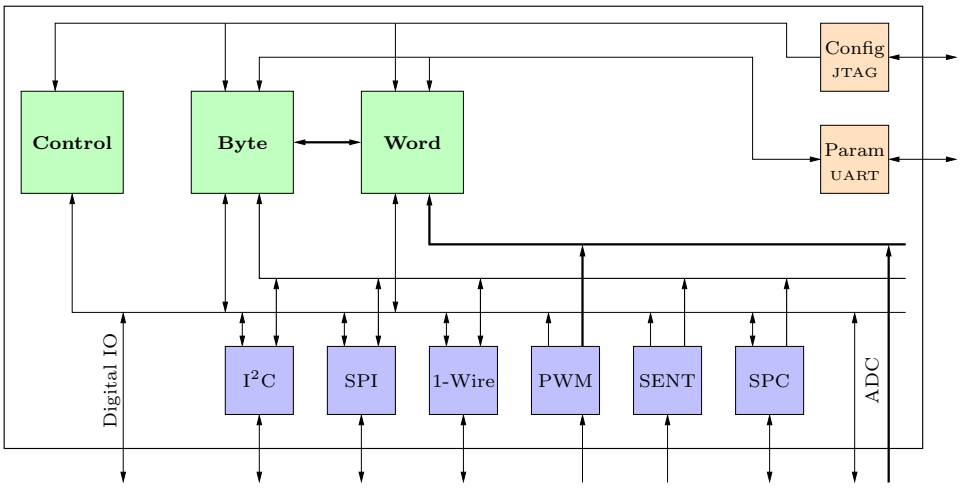


Fig. 3. Structure of the test chip.

4.2 Test Chip

The test chip (see Fig. 3) is a model for one reconfigurable module (green). This is supplemented by peripherals (blue) and the configuration infrastructure (orange). The reconfigurable module is built of three parts: control logic, byte registers and word arithmetic unit. The control logic is dedicated to set and react on digital signals and contains four TR-FSM instances. The byte registers store byte-wide (8 bits) values and the word arithmetic unit (16 bits) is used for e.g. integer comparisons. The chip includes 16 digital inputs and 16 digital outputs. To simulate the sensor measurements an ADC interface and serial bus masters (I^2C, SPI, 1-Wire, PWM, SENT and SPC) are included to connect an external sensor. All these units and peripherals are connected with control, byte and word signals where applicable. The byte-oriented peripherals are the I^2C, SPI, 1-Wire, SENT and SPC bus masters while the PWM and ADC are word-oriented peripherals. Note that there is no address and data bus but just directly connected signals.

All TR-FSM instances in the *control unit* have varying size and are connected via multiplexers and demultiplexers to the high number of input and output signals.

Byte oriented peripherals (e.g. the I^2C bus master) are connected to the *byte unit*. Its purpose is two-fold: When e.g. the external I^2C sensor is queried, first a special sequence of bytes (address, control, ...) is written to the sensor. These are taken from the previously setup byte registers in the byte unit. Then the sensor value is read byte-wise and stored to other byte registers. These have a special multiplexer to forward this value to the word unit.

The *word unit* is a heterogeneous collection of word-wide functional units like an absolute difference ($|A - B|$), a configurable adder/subtracter, a counter, comparators and registers. These are connected via configurable multiplexers to input signals and output signals as well as a feedback path. The elements are also equipped with bit signals, e.g. the carry output of the adder, a comparison result or a strobe signal for registers.

The *configuration* is implemented as long shift-register chains. This allows to connect several configurable elements in series (e.g. all elements of a TR-FSM). To limit the length of these chains and to allow individual access to independent parts of the chip, multiple instances of such chains are implemented. Independent chains are implemented for every TR-FSM instance, the input and output multiplexers as well as the internals of the control, byte and word units. The configuration procedure is performed similar to the download of the config bit stream of an FPGA via a JTAG interface. Every config chain can be selected as data register, so the bits are shifted through this chain. If a TR-FSM is implemented on a SoC together with a (master) CPU, the configuration interface would ideally be implemented as memory mapped registers to be accessed by the CPU. The configuration interface should then automatically serialize the data and shift into the config bit chain.

Contrary to the configuration, the *parameterization* is used to set byte values (two consecutive bytes can constitute a word). This is implemented as an addressable region with a simple protocol transferred via an UART (RS232) interface. Only the byte and word units have parameterizable values. The same interface is also used to query values, e.g. a sensor value stored in the word unit.

4.3 TR-FSM Implementation

The TR-FSM was described in VHDL and synthesized with Synopsys Design Compiler. For chip layout the tool Magma Talus was used. The number of input and output signals, the width of the state vector as well as the number of TRs and their width are configured with generics.

In the main unit the TRs are instantiated as specified by the generics and connected with input and state signals. The next state and output value are fed to two large multiplexers. Note that this needs a one-hot encoding for the select signal instead of a binary encoding, since each TR has its own match flag output. This is used to mask off the inactive next state values which are finally ORed across the full height.

The state register is fed with the output of the next state multiplexer and stores the new value on the next rising clock edge. Its reset value is the `0`-vector. The second large multiplexer for the output value directly outputs its value as the TR-FSM output.

All parts of the TR are purely combinational. Each holds an SSG which outputs a logic `1` if the current state is equal its configured value. This activates the IPG which receives the input signals selected by the ISM according to its configuration. The IPG creates the match output of the TR. Besides these active parts, every TR holds the NSR and OPR as passive parts to offer the configured next state and output pattern. Additionally TRs without inputs are implemented, which simply activate the next state for an unconditional linear sequence These simply forward the output of the SSG as the match output.

As discussed in Sec. 3.1.3 the ISM is implemented with a reduced number of switches. The configuration is applied in boolean encoding, so no optimization of the configuration length is done. The implementation uses a combination of AND and OR gates to distribute the configuration bits and input signals across the output signals. While this creates some long paths with cascaded gates, this does no harm because these paths are only related to configuration values, which don't

Table I. TR-FSM instance parameters

Instance	n_I	n_O	n_S	n_T 0	1	2	3	4	5	T
1	5	5	4	5	5	5	4	3	0	22
2	10	10	5	8	8	8	8	6	2	40
3	10	10	5	8	8	8	8	6	2	40
4	15	15	5	14	12	12	11	8	4	61

Table II. TR-FSM instance synthesis results. The area is given in μm^2, the cell count is split in combinational and sequential cells.

Instance	n_I	n_O	n_S	T	Area	Cell count comb.	seq.	total
1	5	5	4	22	29,900	891	485	1,376
2	10	10	5	40	94,800	3,347	1,397	4,744
3	10	10	5	40	94,600	3,382	1,397	4,779
4	15	15	5	61	187,800	7,193	2,651	9,844

change often. The real signals are conducted through one AND plus one OR gate. The IPG (see Sec. 3.1.2) is implemented as a lookup-table.

All configurable parts are enclosed in the TRs which connects the configuration chain from the SSG, via the ISM, IPG and NSR to the OPR. All TRs are also connected in series from the lowest width to the highest width.

Table I shows the parameters of the four TR-FSM instances. These were chosen empirically by respecting the available chip area and some example configurations. Two instances intentionally have the exact same parameters.

4.4 Chip Area

The test chip was synthesized for a 130 nm 6 metal layer standard cell library and produced by Infineon Technologies. For optimization the hierarchy was flattened. A total core area of 1,100 μm by 940 μm = 1,034,000 μm^2 was available of which 927,128 μm^2 were used, the rest was filled with filler cells, i.e. the utilization is 89.7 %. From the operating cells a total of 486,836 μm^2 cells are combinational and 440,342 μm^2 are sequential.

TR-FSM Totals. The locations of the four TR-FSM instances on the chip core area are shown in Fig. 4. Table II summarizes the synthesis results for the chip area and number of cells. It clearly shows that the number of inputs n_I has a significant influence on the total area, as will be detailed in the next section. Note that the instances 2 and 3 have identical parameters but the tools optimized them differently, so they show slightly different numbers of cells. For all further evaluation of the TR-FSM instances, the values of instance 2 and 3 are averaged.

Proportions. An evaluation of the proportion on area consumption of the individual building blocks of an TR-FSM (see Fig. 5) shows, that the TRs (SSG, ISM, IPG, NSR, OPR) account for 91.3 %, 92.9 % and 93.7 % of the area for the instances 1, 2 and 3, and 4, respectively. The ISM and the OPR grow disproportionate due to the strong dependency on the number of inputs n_I and outputs n_O, respectively: All ISMs of TR-FSM instance 1 require 20 % of the area, which increases to 31.2 % for instances 2 and 3 and to 37.8 % for the largest instance 4. The relative OPR size increases from 16.6 % to 19.5 % and 22.4 %.

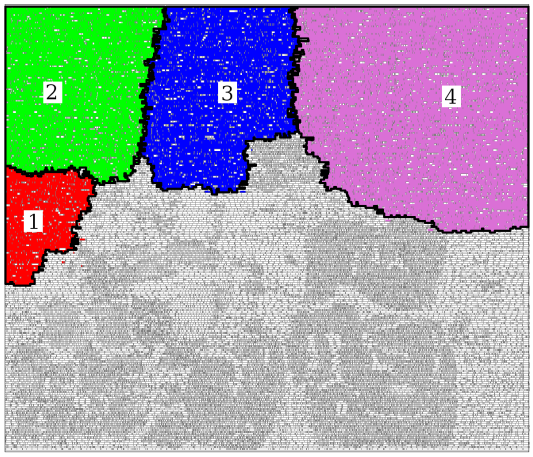


Fig. 4. TR-FSM instances on the testchip marked with colors red (1), green (2), blue (3) and violet (4).

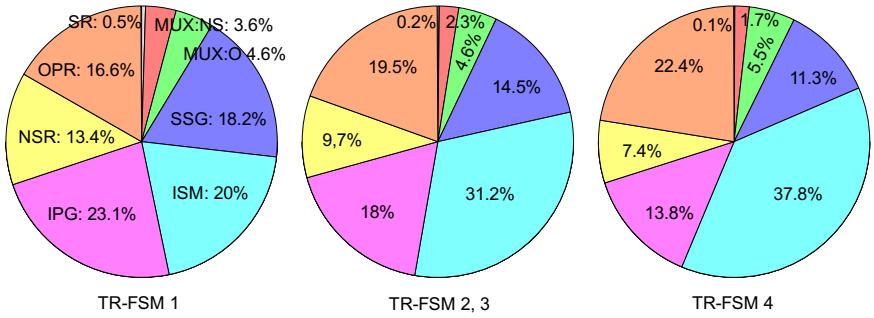
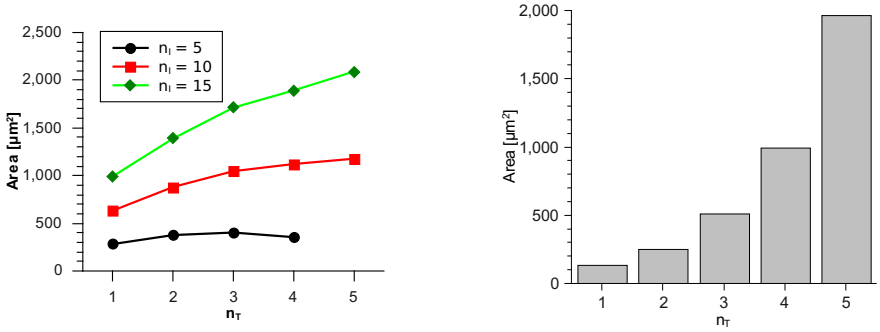


Fig. 5. Area proportions of the TR-FSM parts, shown for the three instance types. MUX:NS: large multiplexer to select next state, MUX:O: large multiplexer to select the output.

Building Blocks. The state register requires considerably less than 1 % of the total area because it is just n_S D flip-flops. The large multiplexers for the next state and the output value (denoted by "MUX:NS" and "MUX:O" in Fig. 5) linearly depend on the number of state bits n_S and outputs n_O, respectively, as well as the number of transition rows T.

Due to the optimized number of switches of the ISMs (see Sec. 3.1.3) the area depends less-than-linear on the number of ISM outputs n_T (see Fig. 6(a)). The graph for $n_I = 5$ shows that for $n_T \gg \frac{1}{2}n_I$ the size even decreases. On the other hand, the more-than-linear dependence on the number of primary inputs n_I is responsible for the disproportionate growth mentioned above.

Fitting an exponential curve to the area of an IPG shows that it exponentially depends on the number of input bits n_I with a mantissa of 1.98 (see Fig. 6(b)). One more input increases the area by a factor of approx. 2. Therefore we can conclude that the main area is consumed by the D flip-flops to store the configuration values



(a) Area of an ISM depending on the number of its outputs n_T. The parameter is the number of inputs n_I.

(b) IPG area depending on the number of inputs n_T.

Fig. 6. Area of ISM and IPG.

of the lookup-table. This clearly shows that larger IPGs should not be implemented as LUTs.

The size of the SSG, NSR and OPR linearly depends on the number of state bits n_S and output signals n_O, respectively, and on the number of transition rows T.

Comparison. Comparing the area consumption of the TR-FSM instances with FPGA instances is difficult because most FPGA vendors refuse the necessary information. For a fair comparison the same semiconductor process (130 nm) should be used, e.g. the Xilinx Virtex-II FPGA [Davis et al. 2002]. Unfortunately the only available information to us was found for the SiliconBlue iCE65 Ultra Low-Power mobileFPGA Family [SiliconBlue Technologies Corporation 2010b], which is produced in a 65 nm technology. The product is available as bare die [SiliconBlue Technologies Corporation 2010a] for which the core logic area can be estimated to 6.7 mm^2 (iCE65L04). Under the assumption that all special purpose logic like JTAG and IO pad drivers are located in the pad frame, the core area is made up of 3520 logic cells (one 4-LUT plus a flip-flop) and 20 RAM4k, where a RAM4k spans 16 logic cells. Then the area of one logic cell is approximately 1750 μm^2. To overcome the technology difference we scale this area by a factor of $\left(\frac{130\,\text{nm}}{65\,\text{nm}}\right)^2 = 4$ which results in 7000 μm^2 per logic cell.

[Bukowiec 2008] synthesized various predefined FSMs from the LGSynth93 benchmark suite [McElvain 1993] for an FPGA architecture with 4-input LUTs and documents the number of used LUTs. From these 39 FSMs we filtered those fitting into the three different TR-FSM instances. In each group the FSM which uses most FPGA LUTs was chosen and the approximate area for the logic cells of the iCE65 FPGA scaled to a 130 nm process was calculated (see Tab. III). For all of those example FSMs the TR-FSM uses considerable less area (25.5 - 32.7%) than an FPGA implementation.

4.5 Timing

The area and gate count of the TR-FSM instances can only be determined with the chip design tools. While such tools also support to estimate or simulate the timing and delay information as well as power consumption, a real measurement offers

Table III. Comparison of chip area required by FSMs using a iCE65 and a TR-FSM.

Instance	FSM	LUTs	iCE65 Area	Scaled Area	% TR-FSM
1	dk15	14	24,500 μm^2	98,000 μm^2	30.5%
2,3	s386	53	92,750 μm^2	371,000 μm^2	25.5%
4	cse	82	143,500 μm^2	574,000 μm^2	32.7%

Instance	ISM	IPG	MUX:O	total
1	1.28 ns	2.94 ns	2.61 ns	6.83 ns
2,3	1.86 ns	3.27 ns	3.15 ns	8.28 ns
4	1.45 ns	4.37 ns	4.05 ns	9.87 ns

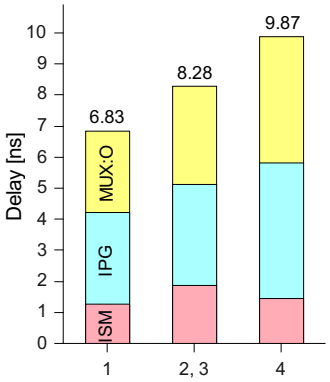


Fig. 7. Combinational delay from input to output of the three TR-FSM instance types. MUX:O: large multiplexer to select the output.

more realistic values. On the other hand, both measurements are confounded by all other parts of the chip, e.g. the IO pads, wiring to the specific area of interest and IO multiplexers. De-embedding the circuit of interest would then introduce additional uncertainty. Therefore we chose to determine the timing information using static timing analysis of the design tools.

The analysis was performed with the worst case operating condition (lowest supply voltage, highest temperature, slowest process corner). Table and Fig. 7 summarize the maximum combinational logic delay (critical path) from the TR-FSM inputs to the outputs, divided into the three stages ISM, IPG and output multiplexer.

The main contributors to the delay are the IPG and the output multiplexer. For the smallest TR-FSM instance the ISM only contributes with two logic gates (1.28 ns). Although there exist paths with a higher number of gates, in the visualized maximum delay case the wiring delay contributes by a larger degree and therefore additional gates rather reduce the delay due to added drivers. The IPG requires 8 gates and 2.94 ns and the output multiplexer has a depth of 5 gates which add a delay of 2.61 ns. The maximum delay case for the middle-sized TR-FSMs passes through a 4-input IPG, despite they have two 5-input IPGs. Its ISM has 4 gates (due to the OR-tree to combine 10 inputs), the output multiplexer contributes another 6 gates. The critical path of the largest TR-FSM instance passed 3 gates through the ISM (again high wiring delay), 9 gates through a 5-input IPG and 6 gates through the output multiplexer.

The comparison of these timing values to an FPGA platform is again difficult due to semiconductor process differences. To overcome this we assume an FPGA with 4-input LUTs which resemble the 4-input IPG. This introduces a delay of

Table IV. TR-FSM current measurement results.

Instance	Total	ISM	IPG	MUX:O
1	3.04 μA/MHz	0.89 μA/MHz	1.14 μA/MHz	1.00 μA/MHz
2,3	9.50 μA/MHz	3.78 μA/MHz	3.65 μA/MHz	2.07 μA/MHz
4	18.3 μA/MHz	9.50 μA/MHz	5.53 μA/MHz	3.29 μA/MHz

2.94 ns to 3.27 ns, i.e. the TR-FSM delay is comparable to two to three stages of such LUTs, which is a typical path length for an FSM. In an FPGA considerable additional delay will be introduced by the flexible routing resources. Therefore the TR-FSM implementation even requires less delay than an FPGA implementation.

Timing measurements with the produced test chip showed a propagation delay time of $t_{\mathrm{pd}} = 15.6$ ns from the input to the output pads of the chip for TR-FSM instance 3. This value contains, besides the TR-FSM, delays from the pads, level shifters and internal wiring and multiplexers. Subtracting the delay of the TR-FSM of 8.28 ns (see Fig. 7), these parts contribute by approx. 7.32 ns, so we assume the calculated delay values above as verified. Since the rise and fall times of the signal are themselves in the range of approx. 12 ns the propagation delay time measurements are very inaccurate. Therefore no further timing measurements were performed.

4.6 Power Measurement Results

For the power measurement the test chip was supplied with $V = 1.2$ V. To investigate the behavior of the four TR-FSM instances, special configuration bit streams were designed. To de-embed the power consumption of the TR-FSM, the total chip supply current was measured with successively enabled stages. The TR-FSM input signal was an alternating "`000...00`" and "`111...11`" pattern which poses the maximum possible change and therefore maximum switching current. This signal was applied with an update rate of 32.768 kHz to 16.67 MHz.

With a linear regression the leakage current was separated from the switching current. The linear model showed an excellent fit, i.e. the current linearly depends on the frequency. Table IV summarizes the switching current in μA/MHz. The total current is further divided in the proportions of the ISMs, IPGs and output multiplexer. Note that this sub-division is somewhat inaccurate due to synthesis optimization. For example, the switching activity of a disabled ISM stops at points distributed throughout the logical depth of the ISM instead of exactly at the last gate.

In Glaser et al. [2009] the current consumption of several ultra low-power microcontrollers was evaluated. The lowest power consumption was 1.20 mA at 3 V and 4 MHz resulting in a normalized current of 100 μA/MHz/V. Compared to the TR-FSM instances (1: 2.53 μA/MHz/V; 2,3: 7.92 μA/MHz/V; 4: 15.3 μA/MHz/V) these require 39.5, 12.6 and 6.6 times less current, respectively. But note that a TR-FSM directly performs the tasks while microcontrollers require a huge overhead (interrupt, context save/restore, peripheral access with memory mapped registers). Therefore, for the same task a large number of clock cycles is spent. This clearly shows that the use of a TR-FSM largely reduces the power consumption.

5. SUMMARY

This paper presents a novel, reconfigurable architecture for finite state machines. It is based on the state transitions instead of the state transfer functions, which is beneficial if the FSMs in question have relatively few transitions. Compared to (embedded) FPGAs the following characteristics of the TR-FSM architecture are beneficial for low power applications.

—The logic function implemented by an FPGA is assembled of an accumulation of fine-grained basic structures. These provide flexible means to implement a virtually infinite number of functions. On the other hand the given basic elements (slices, look-up tables, adaptive logic modules, ...) don't fit all requirements and thus are either used only partially or have to be combined in a complex manner. Both cases mean high overhead and/or unused resources, which lead to increased power consumption and area requirements compared to the TR-FSM structure.

—The basic logic elements of FPGAs are connected via rich and flexible routing resources. The involved short and long wires impose a high capacitive load and include numerous pass transistors along the path. Both result in increased delay, power consumption and area overhead. The TR-FSM architecture uses direct connections between the optimized locations of the logic gates.

—The unused transition rows of an TR-FSM can be switched off by voltage gating for power saving. This is not possible for FPGAs.

Other advantages are the constant latencies of the TR-FSM, since the routing is fixed, and the fact that the synthesis of an FSM to a specific TR-FSM is very easy, since arbitrary state encodings can be used and no sophisticated minimization problems are involved.

The evaluation above also showed some drawbacks, which primarily stem from the utilization of a standard cell library. The architecture contains multiplexers (ISM, next state, output) with a high number of inputs. These are synthesized to trees of logic gates which increases the delay by $o(\mathrm{ld}\, n)$ and area by $o(n \,\mathrm{ld}\, n)$. FPGA architectures tackle this by the use of transmission gates. A similar approach was investigated using heterogeneous trees [Alioto et al. 2002] and incorporating interconnect parasitics [Alioto and Palumbo 2007b; 2007a]. Unfortunately in a semi-custom design flow such elements are not available and require a big effort to provide models for timing evaluation and driver strengths for the synthesis tools. The development of such macros is planned for the next implementation.

A real implementation of the proposed architecture showed a considerable reduction in area, delay and power compared to FPGA architectures.

A further extension are multi-bit LUTs for the IPG, such that a single transition row can handle transitions to multiple different states and/or with varying output patterns, with only a little overhead in the LUT. For linear sequences specialized time-out transition rows with an integrated counter are considered.

Transitions which have to consider a high number of input signals should be mapped to special transition rows with IPGs implemented as sum-of-product logic. Another option is to combine two transition rows and place a MUX to select between the outputs of the two IPGs. Its select input is then driven by another FSM input, so increasing the total input signal sensitivity list by one. Finally, by enhancing

the NSR to latch the current state, sub-state-machines which are able to return to the caller may be implemented.

REFERENCES

Alioto, M., Di Cataldo, G., and Palumbo, G. 2002. Optimized design of high fan-in multiplexers using tri-state buffers. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 49,* 10 (Oct.), 1500–1505.

Alioto, M. and Palumbo, G. 2007a. Design of Fast Large Fan-In CMOS Multiplexers Accounting for Interconnects. In *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS*. 3255–3258.

Alioto, M. and Palumbo, G. 2007b. Interconnect-Aware Design of Fast Large Fan-In CMOS Multiplexers. *IEEE Transactions on Circuits and Systems II: Express Briefs 54,* 6 (June).

Bukowiec, A. 2008. Synthesis of Finite State Machines for Programmable Devices Based on Multi-Level Implementation. Ph.D. thesis, University of Zielona Gora, Poland. http://zbc.uz.zgora.pl/Content/14528/PhD-ABukowiec.pdf (retrieved 2009-11-03).

Davis, S., Reynolds, C., and Zuchowski, P. 2002. IBM Licenses Embedded FPGA Cores from Xilinx for Use in SoC ASICs. Tech. rep., Xilinx, Inc.

Glaser, J., Damm, M., Haase, J., and Grimm, C. 2010. A Dedicated Reconfigurable Architecture for Finite State Machines. In *Reconfigurable Computing: Architectures, Tools and Applications, 6th International Symposium, ARC 2010*. Lecture Notes on Computer Science, vol. LNCS 5992. Springer, Bangkok, Thailand, 122–133.

Glaser, J., Haase, J., Damm, M., and Grimm, C. 2009. Investigating Power-Reduction for a Reconfigurable Sensor Interface. In *Proceedings of Austrochip 2009*. Graz, Austria.

Glaser, J., Haase, J., Damm, M., and Grimm, C. 2010. A Novel Reconfigurable Architecture for Wireless Sensor Networks. In *Tagungsband zur Informationstagung Mikroelektronik 10*. OVE, Vienna, Austria, 284–288.

Glaser, J., Haase, J., and Grimm, C. 2010. Designing a Reconfigurable Architecture for Ultra-Low Power Wireless Sensors. In *Proceedings of the Seventh IEEE, IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. Northumbria University, Newcastle upon Tyne, United Kingdom, 343–347.

Hartenstein, R. 2001. Coarse Grain Reconfigurable Architecture. In *Proceedings of the 2001 Conference on Asia South Pacific Design Automation*. Yokohama, Japan, 564–570.

Katz, R. H. 1994. *Contemporary Logic Design*. The Benjamin/Cummings Publishing Company, Inc.

Liu, Z., Arslan, T., and Erdogan, A. T. 2006. An Embedded Low Power Reconfigurable Fabric For Finite State Machine Operations. In *International Symposium on Circuits and Systems, ISCAS*. 4374–4377.

Liu, Z., Arslan, T., Khawam, S., and Lindsay, I. 2005. A High Performance Synthesisable UnsymmetricaI Reconfigurable Fabric For Heterogeneous Finite State Machines. In *Proceedings of the ASP-DAC*. Vol. 1. 639–644.

McElvain, K. 1993. LGSynth93 Benchmark Set: Version 4.0. online: http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth93/.

Milligan, G. and Vanderbauwhede, W. 2007. Implementation of Finite State Machines on a Reconfigurable Device. In *Proceedings of the second NASA/ESA Conference on Adaptive Hardware and Systems*. Edinburgh, 386–396.

Rabaey, J. M., Chandrakasan, A., and Nikolic, B. 2003. *Digital Integrated Circuits*. Prentice Hall, Upper Saddle River.

SiliconBlue Technologies Corporation 2010a. *DiePlus Advantage*. SiliconBlue Technologies Corporation.

SiliconBlue Technologies Corporation 2010b. *iCE65 Ultra Low-Power mobileFPGA Family*. SiliconBlue Technologies Corporation.

Received September 2010; revised November 2010; accepted Month Year