A POWER EFFICIENT REGISTER FILE ARCHITECTURE USING MASTER LATCH SHARING

M. Wróblewski[1], *M. Mueller*[1,2], *A. Wortmann*[2], *S. Simon*[2], *W. Pieper*[2], *J. A. Nossek*[1]

[1] Munich University of Technology
Marek.Wroblewski@ei.tum.de

[2] Hochschule Bremen
Matthias.Mueller@hs-bremen.de

ABSTRACT

This paper introduces a method of reducing area and power consumption of a synthesizable register file by using a single master latch shared by a number of slaves. It investigates potential timing problems and discusses possible solutions. Presented simulation results show that, depending on the size of the register file, reduction of power consumption of more than 50% is achievable.

1. INTRODUCTION

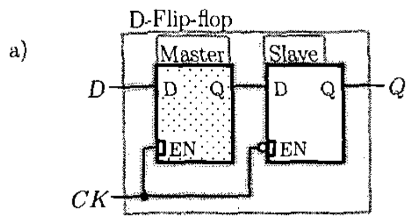
The reduction of power dissipation in modern VLSI circuits is one of the most important challenges. With increased die size and advanced process technologies, whole systems with an ever-growing number of transistors are integrated on a single chip. The problem of heat dissipation in low cost packages of high performance systems [4] and a growing demand for portable consumer products with an independent power supply drive the development and application of low-power design techniques [1].

Data stores are an important power critical part of resource sharing architectures [2] or processing units, like application specific instruction set processors (ASIPs). They are preferably implemented as a synthesizable register file described as part of the design on register transfer level, because of a high effort required for timing verification of RAM.

Therefore, in this paper we consider data stores implemented in this manner. In general, they consist of a number of word level registers. The data inputs of the registers are connected to the data bus and represent a high capacitive load. This results in a high power consumption per signal transition. While the number of unnecessary transitions can be kept low by integration of glitch barriers into the signal path, further improvements can be achieved by reduction of the capacitance of the data bus, e. g. as proposed in [3]. A rearrangement of the registers in the register file is performed, reducing the number of registers connected directly to the data bus.

This paper presents a method which also aims at reduction of capacitance connected to the data bus. This is achieved by splitting up the master-slave flip-flops into the master latches and the slave latches. If clock gating is applied, slave latches of registers in the register file can share one master latch. Thus the number of master latches connected to the data bus is decreased. Additionally savings in area can be expected.

The paper is organized as follows: In Section 2, the modified register file is derived and explained. Section 3 addresses timing-related issues involved in application of this method. In Section 4 experimental results are presented.



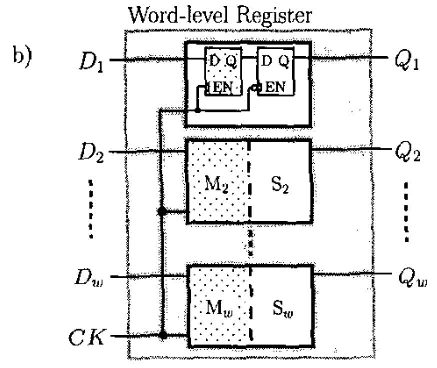


Figure 1: a) Master-slave D-flip-flop. b) Word-level register.

2. MODIFIED REGISTER FILE

Storage elements found in libraries of semicustom technology providers are frequently realized as edge-triggered master-slave D-flip-flops as shown in Figure 1 a). They consist of a master and a slave latch. If a value is to be stored into the flip-flop, the master latch freezes the value with the triggering edge of the clock signal and the slave latch becomes transparent. The value is visible at the output of the flip-flop. At the other edge of the clock signal, the slave latch stores the value and the master latch gets transparent. For data stores, w flip-flops are grouped to word-level registers, where w denotes the number of bits of the data signal. Each bit of the data signal is connected to one flip-flop of the word-level register, as shown in Figure 1 b). Register files, in which r data values can be stored, consist of r word-level registers, as shown in Figure 2 a). Since the data inputs of all registers are connected to the data bus this leads to a high bus capacitance.

Two approaches are commonly used to update data stored in registers. First, a feedback loop from the output is connected via a multiplexer to the input of the flip-flop. If a new value is to

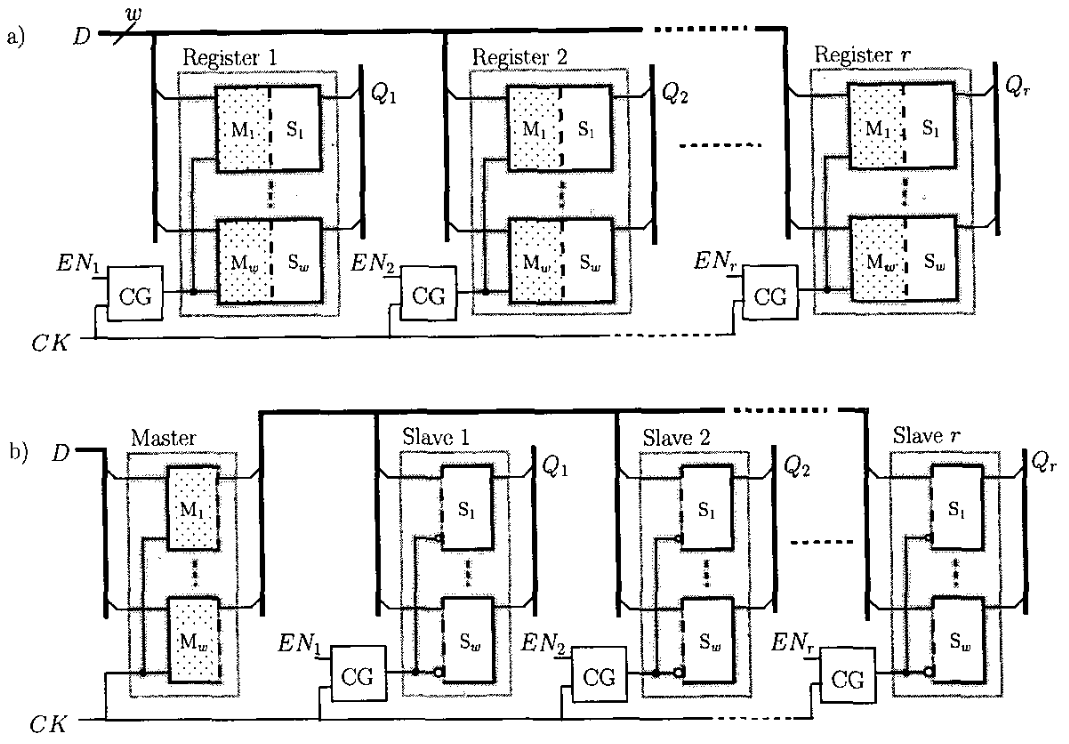


Figure 2: a) Conventional register file with flip-flops. b) Modified register file with shared master latches.

be stored it is routed via the multiplexer to the input. Otherwise the output of the flip-flop is connected to the input and the stored information is rewritten in every clock cycle. Additional enable logic is responsible for controlling the multiplexer.

The second approach is the application of clock gating. Due to significantly reduced power consumption this is usually preferred and in the following we only consider this case. This method was applied to the register file shown in Figure 2. Here, logic and possibly sequential cells (represented as blockCG in Fig. 2) are inserted into the clock path. The same enable logic as in the previous case is used, this time however to decide whether to clock a given register. Thus data is written only when it changes and clock signals of not selected registers are disabled.

As consequence however every master latch is transparent all the time except when new data is written into its slave. Every transition on the data bus is repeated by all master latches, although most of them (usually all but one) do not need to pass the new state to their slaves. This requires that internal capacitances of these latches (and input capacitances of slaves) be recharged and increases power consumption.

In the following we limit our investigations to architectures where only a single piece of data can be stored per clock cycle. It may be stored into many registers simultaneously, but all registers with clock signal enabled receive the same value.

Under this assumption it is possible to replace all r master latches connected to a specific bit-line of the data bus by one single master latch. The corresponding slave latches share this master latch. This is shown in Figure 2 b).

This configuration has the following advantages:

- Since there is only one master latch connected to the data bus, the capacitive load is reduced. Hence, a reduction of power dissipation in driving modules can be expected.
- Transitions seen on the data bus cause only the internal capacitances of a single latch to be recharged, and this only during second half of clock period (the shared master latch is clocked in every clock period). This reduces power consumption of the register file.
- Less area is required for the register file (and possibly for the driving blocks).

3. TIMING

Obviously the reduction of power consumption comes at a cost. By splitting up the master and slave latches of a flip-flop we give up the main advantage of the foundry-designed flip-flop – the well defined timing conditions. The master and the slave placed in a single cell are adjacent and any clock signal delays can be compensated when designing the cell. When using clock gating in this scenario both the master and the slave are clock gated.

It is not so in the proposed approach. We only apply clock gating to the slave latches, as master is shared and operates in every clock cycle. As the slaves' clock signal is delayed by the clock gating cells it cannot arrive at the clock inputs of the slaves at the same instant as the clock signal of the master arrives at its clock input. This may lead to serious functionality problems as we show later.

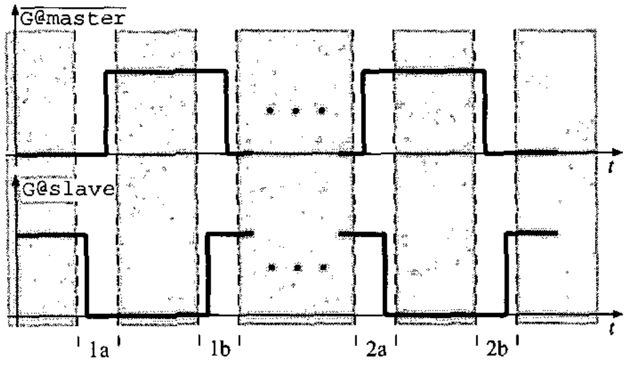


Figure 3: Enable signals of master and slave latches of a split-up flip-flop.

Secondly we have virtually no control over the physical placement of the components of a single flip-flop. In a large design the delays introduced by careless placement may render the design unusable. We discuss this issue in section 3.2.

3.1. Clock gating related timing issues

Let us first evaluate the effects of the delay introduced by clock gating cells.

For sake of the following discussion let us assume that we use rising-edge triggered flip-flops and their slave latches are transparent when their enable signals G are low. We consider the following cases:

1. $\overline{\texttt{G@master}}$ follows `G@slave` (cf. left portion of the diagram in Fig. 3).
    (a) Short before the rising edge of $\overline{\texttt{G@master}}$ both latches are transparent and glitches of the data signal appear at the output of the flip-flop. However, we do not deem this a serious functionality issue if the setup-time condition of the flip-flop is set appropriately. With the rising edge of $\overline{\texttt{G@master}}$ the value of the data signal is captured by the master latch and the output of the flip-flop does not change.
    (b) After the falling edge of $\overline{\texttt{G@master}}$ the master latch becomes transparent again. The slave however has already been shut for some time and the value from the previous clock cycle is safely latched.
2. `G@master` leads `G@slave` (cf. right portion of the diagram in Fig. 3).
    (a) At the time when $\overline{\texttt{G@master}}$ changes to high the value of the data signal is stored into the master latch. `G@slave` remains high for some time to come and after it becomes low, the value is visible at the output of the flip-flop.
    (b) After $\overline{\texttt{G@master}}$ is set to low both latches are transparent until `G@slave` becomes high again. If during this time a signal change occurs at the input of the master it will be passed down to the slave and incorrectly latched as soon as `G@slave` goes high. This is the critical case that we address in the following in more detail.

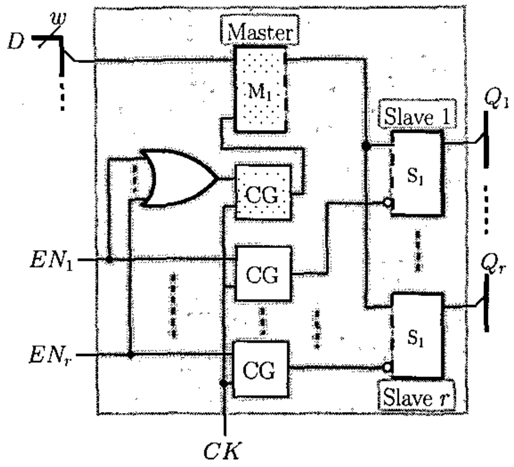


Figure 4: Synchronization of clock signals of master and slave latches.

From the above said it becomes evident that if we can avoid case 2b (i. e. `G@master` leads `G@slave`) then we should be able to guarantee the proper function of the split-up flip-flop. As explained it is not imperative that `G@master` and `G@slave` be perfectly synchronous. In order to achieve this we may delay `G@master` by inserting a quasi clock gating cell as shown in Fig. 4. We use ORed enable signals of the slaves as the enable signal for the master. Assuming modest logic depth of the slaves' enable logic, the latch in the cell is able to suppress any glitching activity generated there, as it is closed during the first half of a clock cycle. The master clock signal is delayed by a simple two-input gate, exactly as is the case with the clock signal of slaves. For these reasons we prefer this solution to one where the clock input of the master latch is connected directly to the output of the OR gate driven by the gated clocks of the slaves.

3.2. Timing issues related to physical placement

While it is not particularly difficult to compensate for delays introduced by well defined blocks, the impact of uncontrolled placement on timing is unpredictable. We consider it therefore necessary to limit the degrees of freedom of the placement and synthesis tools.

This could be achieved by treating every block containing a master latch and all its slaves as an entity. Supplied with necessary information the tool could then optimize the wire delays in a manner such that the conditions postulated in Section 3.1 are satisfied. While this approach offers the designer most flexibility, it currently requires manual intervention during writing HDL code, synthesizing and routing the design.

Taking this idea a step further, one could conceive of a family of library cells, each consisting of a master latch and a fixed number of r slaves, which for sake of brevity we call in the following 1-to-r cells. This number corresponds to register count in the register file. Designing e. g. a register file with 8 registers, 32 bit wide each, the designer would utilize 32 cells, each containing a master

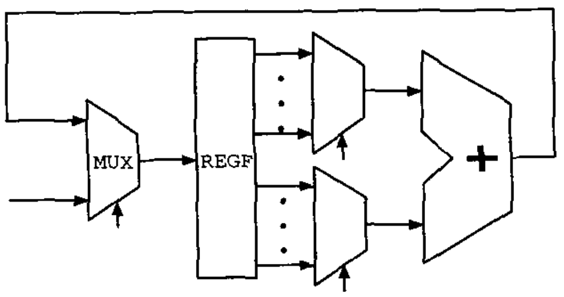


Figure 5: Simulated architecture

and 8 slaves (32 1-to-8 cells).

This kind of cell obviates the need for controlled placement because all the timing requirements are fulfilled internally by the cell and assured at the time when the cell is designed. This closely matches the usual situation when complete flip-flops are used.

The obvious disadvantage of this method is that the designer is limited to a fixed number of registers in the register file. Combining two cells e. g. a 1-to-8 and a 1-to-4 cell in order to create a 1-to-12 cell is not possible without enlarging the cell by additional logic. Even if it were implemented, we would give up the main advantage of the approach – the controlled timing.

Furthermore such a cell crosses the boundaries of logical blocks. Designers usually think in terms of functional blocks as units, e. g. ALU, register file, register, flip-flop, etc. A single 1-to-r cell belongs to many of those blocks, because every slave latch constitutes a bit of a different register. While it might be possible to hide this problem from the designer, modifications to existing synthesis tools may be necessary in order to enable automatic instantiation of this kind of cells.

4. SIMULATION RESULTS

We used the design shown in Fig. 5 to test the impact of the method on the power consumption of the register file. We limited our investigations to this simple architecture because it enables us to study the effects of the transformations without unnecessary overhead in terms of simulation times and complexity, that a more elaborate design would entail.

The design was described in VHDL, synthesized using commercial synthesis tool and a 0.18 μm process standard cell library. All simulation results presented in Table 1 were obtained by power simulations on transistor level and do not take into account any placement or routing information.

For register files containing more than 4 registers power savings can be achieved. As expected, not only the register file itself consumes less power, but also the unit driving register file's inputs (in this case MUX) requires less power due to reduced driven capacitance. Total power savings given in the table are quite high, because the architecture's power consumption is dominated by the register file. In designs containing more elaborate ALU the impact of the application of this method is going to be correspondingly lower. Nevertheless the reduction of power consumption of the register file alone is certainly not to be neglected in power-conscious designs.

r	s	unit	block	8 bit	16 bit	24 bit	32 bit
4	0	μW	MUX	6.22	12.15	17.00	20.34
			REGF	**25.17**	**51.38**	**67.41**	**89.11**
			Total	64.32	137.22	189.43	238.76
	1	-	MUX	0.89	0.85	0.86	0.84
			REGF	**1.03**	**0.94**	**0.99**	**0.94**
			Total	1.00	0.96	0.99	0.96
8	0	μW	MUX	10.55	18.53	28.50	30.82
			REGF	**39.89**	**77.48**	**110.28**	**132.47**
			Total	94.27	188.22	280.00	322.56
	1	-	MUX	0.70	0.64	0.64	0.63
			REGF	**0.75**	**0.70**	**0.67**	**0.70**
			Total	0.85	0.84	0.84	0.84
16	0	μW	MUX	14.51	26.52	37.37	18.45
			REGF	**61.76**	**118.38**	**162.93**	**199.90**
			Total	122.31	239.61	337.44	423.61
	1	-	MUX	0.56	0.49	0.46	1.04
			REGF	**0.63**	**0.57**	**0.55**	**0.55**
			Total	0.76	0.73	0.73	0.69
32	0	μW	MUX	10.15	15.35	20.45	22.43
			REGF	**110.87**	**215.98**	**316.01**	**356.77**
			Total	196.29	391.06	581.77	652.27
	1	-	MUX	1.06	1.01	1.00	1.01
			REGF	**0.52**	**0.44**	**0.41**	**0.42**
			Total	0.63	0.56	0.55	0.55

Table 1: Power consumption for architecture of Fig. 5. **r** denotes the number of registers in the file, **s** indicates whether the flip-flops were split-up (1) or not (0). Dimensionless figures specify the relation of power consumed by the variant in question divided by power consumed by corresponding reference variant.

5. CONCLUSION

The presented method offers a possibility of reducing area and power consumption of the register file in architectures which do not store more than a single value per clock cycle. Although power savings of up to approx. 50% seem promising, some timing-related issues need to be resolved. In future we plan to investigate the solutions proposed in Sec. 3.2 and test their effectiveness by performing placement and routing of our test design. This will also give us an opportunity to confirm the presented power figures and estimate area savings.

6. REFERENCES

[1] A. P. Chandrakasan and R. W. Broderson. Minimizing power consumption in digital CMOS. In *Proceedings IEEE*, page 498, 1995.

[2] K.K.Parhi. *VLSI Digital Signal Processing Systems*. Wiley-Interscience, 1999.

[3] M. Mueller, A. Wortmann, S. Simon, S. Wolter, S. Buch, M. Wróblewski, and J. A. Nossek. Low power register file architecture for application specific DSPs. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages IV/89–IV/92, Scottsdale, USA, May 2002.

[4] G. K. Yeap. *Practical Low Power VLSI Design*. Kluwer Academic Publishers, 1998.