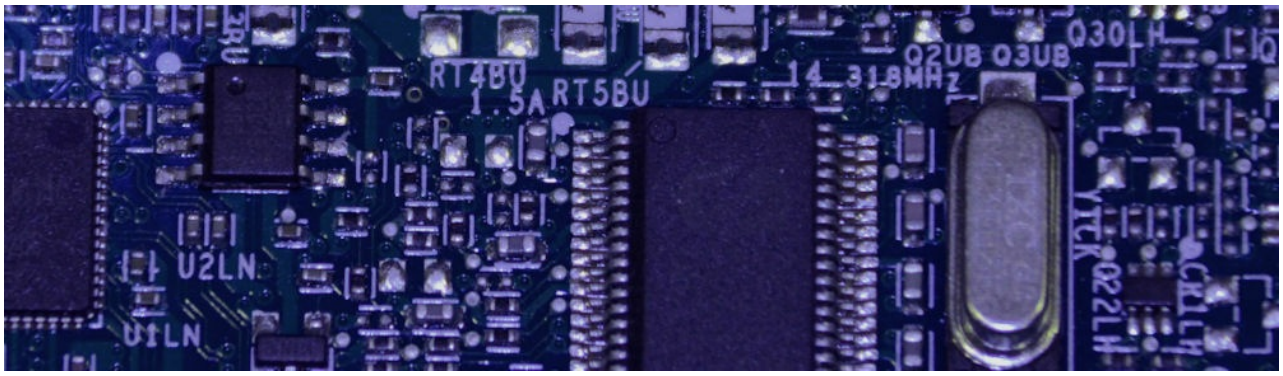
H-bridge Control



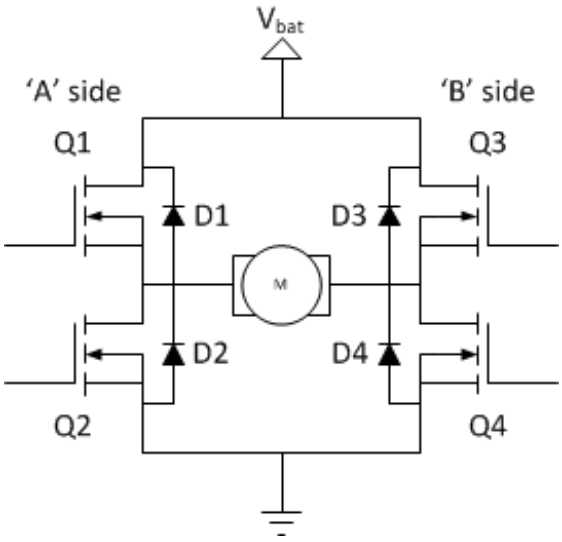
Introduction

In the previous part of the series we went through the various circuits that can take logic level digital signals and make them suitable for driving the gates of the bridge power FETs. These circuits range from trivial to complex, and have some interesting properties, like turn-on and –off delay or limits on duty cycle that will have implications for our current discussion.

In this article we will step further back from the 'bare metal' of the H-bridge and concentrate on the digital control of the bridge. The discussion will concentrate on how to generate the various control signals and how to implement the different drive modes we've discussed before.

For the most part, the detailed implementation of the driver and the bridge itself (PMOS or NMOS devices, etc.) are not a concern here, though occasionally we'll have to take the special requirements of those parts of the design into consideration.

I will use the following h-bridge element notations during the article:



Bridge driver control

As we've seen before there are many drivers and there are many ways those drivers need to be controlled as well. For both low- and high-side drivers, the two basic schemes are active low or active high control.

In active low control, the FET is closed, when the control signal is low, or 0:

control	**FET state**
0	close (conduct)
1	open (doesn't conduct)

In active high control, the sates are the opposite:

control	**FET state**
0	open (doesn't conduct)
1	close (conduct)

With half-bridge drivers, which control one low and one high-side FET, the options are more complicated. Some simply expose two control signals (active low or high) each for the two controlled transistors. Some however have an 'enable' and a 'PWM' pin for example. Again, these can be active low or active high. I will not bore you with all the possible combination (check the datasheet for your part for the exact details) but one example could be the following:

PWM	**enable**	**High-side state**	**Low-side state**

0	0	open (doesn't conduct)	close (conduct)
1	0	close (conduct)	open (doesn't conduct)
0	1	open (doesn't conduct)	open (doesn't conduct)
1	1	open (doesn't conduct)	open (doesn't conduct)

As you can see the enable signal is active low in this example, and the PWM non-inverted.

Another control method is when the driver exposes only a single, three-state control input. When driven low, or high, one or the other FET conducts, when left floating, the both FETs are open:

control	**High-side state**	**Low-side state**
0	open (doesn't conduct)	close (conduct)
1	close (conduct)	open (doesn't conduct)
Z	open (doesn't conduct)	open (doesn't conduct)

Full-bridge drivers can expose an even wider variety of control options, since they drive four FETs. Sometimes they have four control signals for each individual FETs, or would look like two independent half-bridge circuits. Other times, they combine the control signals for the 'A' side and 'B' side of the bridge into fewer control signals. When that happens (like the HIP4080A from Intersil), the driver might start to limit the control patterns and drive modes you can use the bridge with. If you recall, both the 'A' and the 'B' side have three possible states, so a full-bridge can have six possible (static) states. This needs a minimum of 3 digital control signals, so if a full-bridge driver exposes less, you should expect limitations. Yet others (the HIP4080A is a good example for that as well) combine some analog functions to help implement closed-loop control.

You can find the exact control method for your chosen driver in its datasheet.

To simplify the discussion in the following chapter, I will assume that the bridge driver has four independent, active-high input signals, one for each FET. If your driver needs a different type of control, you can easily convert the signals to your particular needs.

Basic drive-mode mapping

Let's first examine, what control states each of the inputs need for the various drive-modes.

Let's start with the lock anti-phase drive in the forward direction:

Lock anti-phase drive, FWD	On-time	Off-time
Q1_CTRL	1	0
Q2_CTRL	0	1
Q3_CTRL	0	1
Q4_CTRL	1	0

For sign-magnitude drive in the forward direction, using high-side off-time conduction we get:

Sign-magnitude drive, FWD, high-side	On-time	Off-time
Q1_CTRL	1	1
Q2_CTRL	0	0
Q3_CTRL	0	1
Q4_CTRL	1	0

The same sign-magnitude drive in the reverse direction:

Sign-magnitude drive, REV, high-side	On-time	Off-time
Q1_CTRL	0	1
Q2_CTRL	1	0
Q3_CTRL	1	1
Q4_CTRL	0	0

Notice so far how the two ‘a’-side control signals (Q1 and Q2) are always inverse of one another. Same is true for the two ‘b’-side drive signals. One would think that we could simply drive them from the same input, through a simple inverter. Unfortunately in practice that would almost always be a bad idea but we will have to come back to that a bit later.

For now, let's finish up the mapping tables for the asynchronous sign-magnitude drive. Here I'll assume high-side off-time conduction as well. For the forward direction we can use the following table:

Async. Sign-magnitude drive, FWD, high-side	**On-time**	**Off-time**
Q1_CTRL	1	1
Q2_CTRL	0	0
Q3_CTRL	0	0
Q4_CTRL	1	0

For the reverse direction:

Async. Sign-magnitude drive, REV, high-side	**On-time**	**Off-time**
Q1_CTRL	0	0
Q2_CTRL	1	0
Q3_CTRL	1	1
Q4_CTRL	0	0

There are some other possible drive patterns that would result in one of these three drive modes. We will see if they are useful in a bit, for now let's just consider these five mappings. By carefully looking at the tables you can see some nice patterns emerge:

- For lock anti-phase drive the 'a'-side control is a mirror image of the 'b'-side one. The PWM signal should be connected to both sides, but in reverse polarity.
- For the phase magnitude drive, two FETs are controlled with a constant signal, only one side is driven with the PWM signal. You can change driving direction by swapping which side gets the PWM drive and which one gets the static signals.
- For asynchronous sign-magnitude drive, only one of the four FETs is switching the other three are at constant voltages. The drive direction is changed by deciding which FET to toggle.

Power-down state

Almost every bridge design needs to have a disabled, or power-down state. There are two possibilities to achieve that. One option is to have at least one of the motor terminals floating. The other is to short the motor terminals together.

For sign-magnitude and asynchronous sign-magnitude drive, this's simple as at 0% duty-cycle that is exactly what happens. For lock anti-phase drive however a new state needs to be introduced for this purpose. Of the many possibilities, let's choose the one where both motor terminals are floating for this discussion:

Lock anti-phase drive	Power-down state
Q1_CTRL	0
Q2_CTRL	0
Q3_CTRL	0
Q4_CTRL	0

In many cases, especially when a microcontroller is in charge of generating the signals for the MOSFETs, it is important that the default power-up state of the bridge is the power-down one. This way, the firmware running on the microcontroller is in no rush of initializing the bridge control signals. It can work on getting the whole system up and running and only after all setup and initialization is done it needs to move the bridge out of power-down into the operational state.

Many microcontrollers start up with their pins in a high-impedance state, when they can't source nor sink current. If those pins are connected to the bridge driver inputs, as long as the firmware doesn't initialize the pins for any other state, the output voltage is not well defined and the wire is susceptible to noise. This can lead to the bridge turning on or worse, getting into a shoot-through condition during startup. To prevent this from happening, its good practice to include pull-up or pull-down resistors on each bridge control pin to make sure they are at a well-defined voltage at all times. The value of the resistors usually isn't critical as long as the don't load the output stages inside the microcontroller too much. It is also good practice to set the resistors up such that they keep the bridge in its power-down state.

Shoot-through protection

We've mentioned shoot-through several times already in previous parts of the series. We've said at the very beginning that the high-side and the low-side switches of an H-bridge on the same side should never ever be turned on at the same time. If that happened, you would create a very low resistance path between your power supply and ground. There are several outcomes of such an experiment and none of them are pleasant. At best, you have some sort of short-circuit protection that trips and your circuit simply looses power. If not, a **lot** of current will start flowing through your circuit. This current will start heating things up and eventually something will break. It will heat up the battery (because of its internal resistance) and overheated batteries can explode. It will heat up the wires which can melt their plastic insulation. It will heat up the PCB

traces, making them de-laminate or even vaporize and destroy the board. It will heat up your FETs and can destroy them as well. You don't want any of these, so you don't want shoot-through. Avoiding static shoot-through is fairly simple: just make sure you never close both 'a' or 'b'-side FETs at the same time.

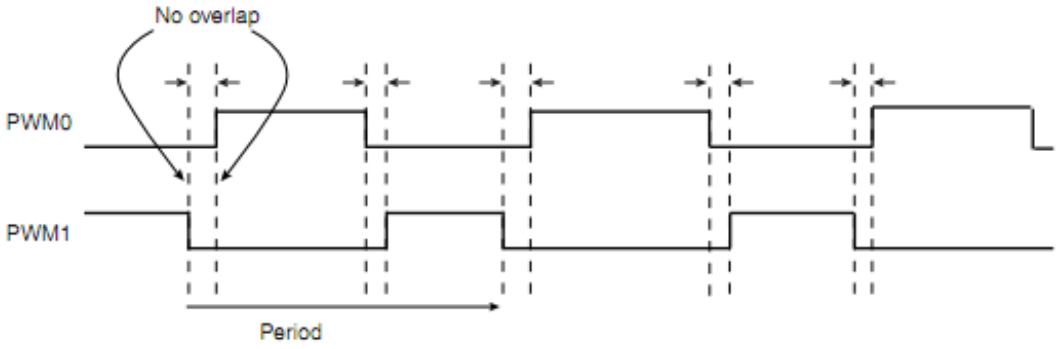
The bigger problem is dynamic shoot-through: when you turn one FET off, while turning the other FET on, for a short while both the low and high-side FETs are potentially conducting to a certain degree, creating a – relatively – low resistance path for the supply to flow to ground. This results in a current-spike that – while not as destructive as static shoot-through – is still quite problematic. **To prevent this, you have to delay the turn-on of the low-side FET by at least as much as the turn-off time of the high-side FET. The same goes of course for the other transition, when you switch from low-side to the high-side.** This technique has many names, dead-time, shoot-through protection, no-overlap PWM, but whatever you call it, unless you know the turn-off times, you can't time this delay properly. This is one of the reasons I've sent so much time in the previous installment talking about turn-on and –off-times and how to calculate and control them.

When you do employ shoot-through protection however, you will end up with a different problem: now, instead of having both the low- and high-side FETs conducting for a short while during the transition, you have none of them conduct for a split-second. At the same time – as we've discussed several times already – the motor current cannot just stop flowing instantly, so some circuit – the catch diodes – will have to take over. As the diodes usually have a higher loss than the switches themselves, during this short time the heat dissipation of the bridge will be higher than normal. This additional dynamic loss on the bridge gets worse as your shoot-through protection window gets a larger and larger portion of the cycle time. So for really high-frequency operations, tight shoot-through protection window control is needed to minimize this source of heat dissipation. In those applications various dynamic shoot-through protection techniques are becoming more and more popular, mostly as part of the bridge driver circuitry.

In many cases, integrated bridge drivers have built-in shoot-through protection circuits with fixed or programmable block-out windows. With those you only have to make sure that your turn-off times are compatible with what the parts can support.

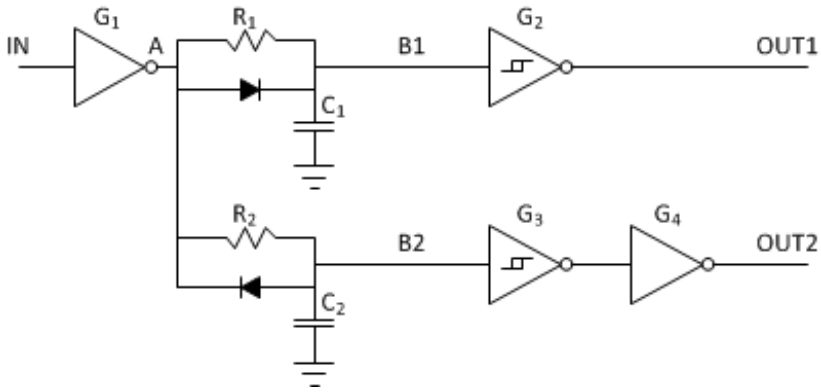
If you are using a part without such protection or are building your own driver, you'll have to make sure that proper shoot-through protection is implemented either in HW or in SW.

Many modern microcontroller PWM implementations give you the option to use two pins that output versions of a single PWM signal, with a programmable no-overlap zone between them, like the SAM7S series from Atmel:

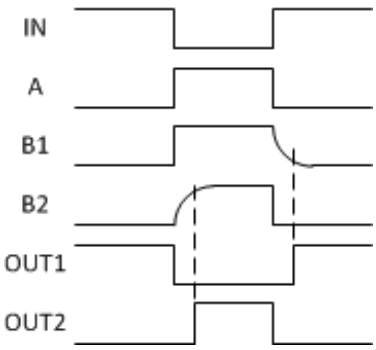


External shoot-through protection circuits

If you need to generate the above wave-forms externally, you can do that for example with the following circuit:



It uses the the R/C circuits to delay the edges of the two outputs, and the diodes to make sure only one of the two edges get delayed. The Schmitt-trigger inverters are needed to clearly define the point where the output switches due to the slow changing of the output of the R/C circuit. The timing diagram of the circuit is the following:



You can see how this circuit reproduces the same non-overlapping outputs that we've seen at the end of the previous chapter. The delay of the falling edge of OUT1 can be calculated as the following:

$t_{dout1} = -R_1 C_1 \ln(V_{tl}/V_{oh})$

where V_{oh} is the high-level output voltage of the input gate, and V_{tl} is the low-level trigger voltage of the Schmitt-trigger. Similarly, the delay of the rising edge of OUT2 is the following:

$t_{dout2} = -R_2 C_2 \ln(1-V_{th}/V_{oh})$

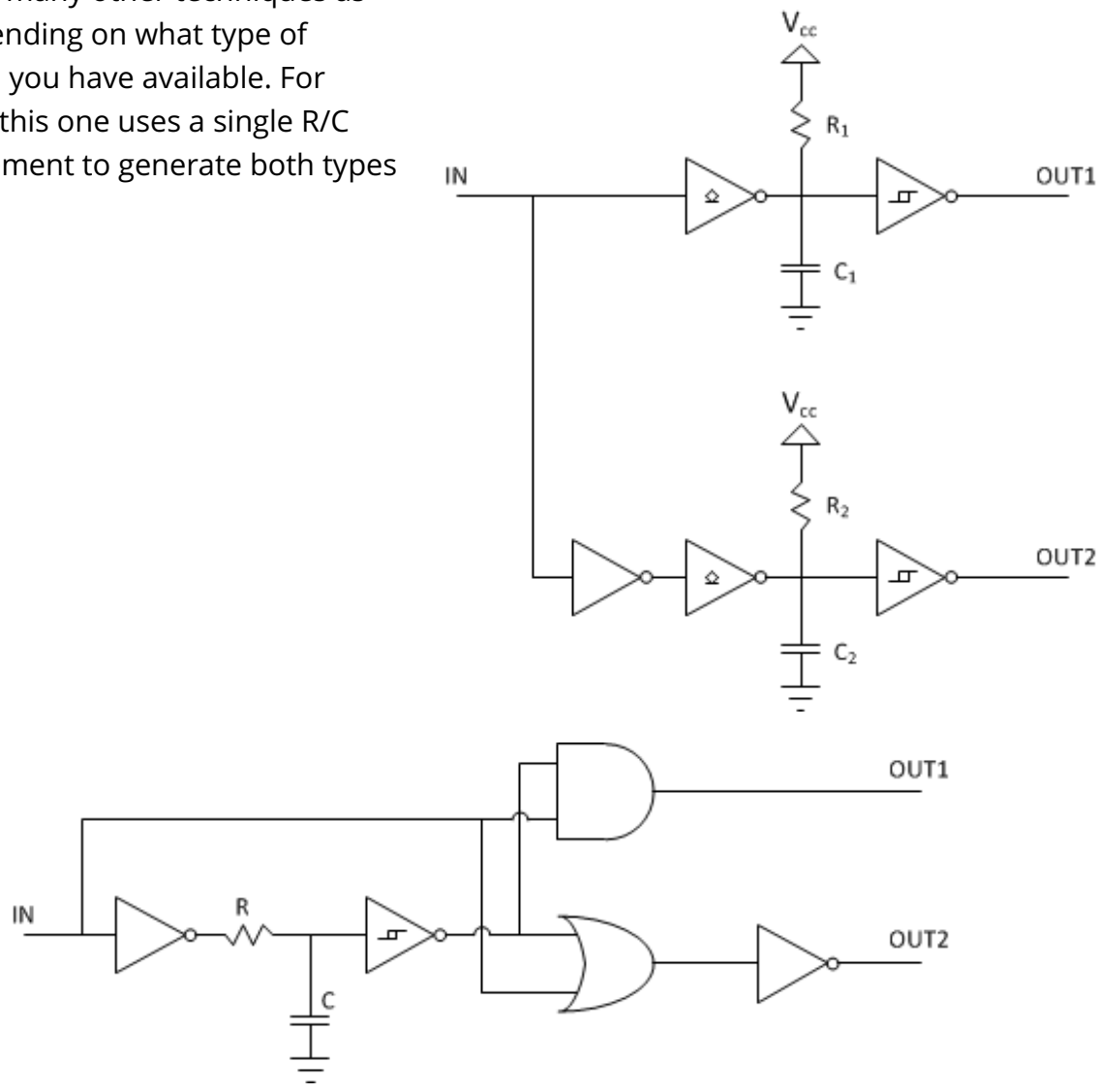
where V_{th} is the high-level trigger voltage of the Schmitt-trigger.

With many Schmitt-trigger circuits having their lower and higher threshold voltages set at 1/3 and 2/3 way between 0 and V_{oh}, both time delays end up roughly R_1 C_1 and R_2 C_2 respectively.

The input gate (G_1) can in many cases be the output driver of the PWM output pin, especially in CMOS logic. Many bridge drivers have Schmitt-trigger input stages so if the input polarity works out the right way, the output buffers (G_2, G_3 and G_4) can be eliminated as well. Even if not, a 6-gate Schmitt-trigger chip, like one of the many variants of the 7414 chip provides a single-chip solution (obviously the normal inverters can also be Schmitt-triggers, they just don't have to be).

Finally, this circuit doesn't produce clear high logic levels at B1 and low logic-levels at B2 due to the forward voltage drop of the diodes. Because of that, a relatively high supply voltage needs to be used, otherwise the signal might not clear both tripping-points of the Schmitt-triggers during their edges. With very low supply voltages, a similar, but different technique using open-drain drivers or discrete FETs can be used to create the asymmetrical delays on the rising and falling edges:

There are many other techniques as well, depending on what type of resources you have available. For example, this one uses a single R/C timing element to generate both types of delays:



PWM routing

Let's re-examine the control patterns in the previous chapter! You've seen that some of the four drive signals are constant during the cycles, some are switching. We can use a simplified notation to describe the various modes. Let's call an external signal **PWM**, which is 1 during the on-time and 0 during the off-time. Another signal, **\PWM** does the opposite. Let's not forget that these signals aren't exactly inverted versions of one another due to the previously mentioned shoot-through protection requirement. Let's also create a signal, **DIR**, which is '1' for forward direction, and '0' for reverse. It's opposite is **\DIR**. Note that when changing direction, the same no-overlap change should be observed on this signal pair as well, so \DIR isn't simply an inverted version of DIR either. With this notation, the drive modes are as follows:

Lock anti-phase drive	
Q1_CTRL	PWM
Q2_CTRL	\PWM
Q3_CTRL	\PWM
Q4_CTRL	PWM

Sign-magnitude drive	**low-side off-time current**	**high-side off-time current**
Q1_CTRL	PWM	DIR
Q2_CTRL	\PWM	\DIR
Q3_CTRL	\DIR	\PWM
Q4_CTRL	DIR	PWM

Asynchronous sign-magnitude drive	**forward, low-side off-time current**	**forward, high-side off-time current**	**reverse, low-side off-time current**	**reverse, high-side off-time current**
Q1_CTRL	PWM	DIR	DIR	DIR
Q2_CTRL	\DIR	\DIR	\DIR	PWM
Q3_CTRL	\DIR	\DIR	PWM	\DIR

Q4_CTRL	DIR	PWM	\DIR	\DIR

You can see that depending on the drive mode, the control signal routing can be fixed or need to depend on the off-time current conduction path chosen (which can be a design-time decision). In the case of asynchronous sign-magnitude drive, the signal routing needs to be dynamic as even a drive direction change changes the PWM signal routing.

This may or may not be problematic, many microcontrollers have flexible enough I/O pins that make all of the modes possible without external logic. However in many cases the support of asynchronous sign-magnitude drive requires four PWM generators inside the microcontroller, as it might not be possible to get a single PWM source appearing on multiple, configurable pins.

Another interesting observation is that asynchronous sign-magnitude drive doesn't use the \PWM signal. As a consequence it doesn't depend on proper shoot-through protection window setting on the PWM output, so if your microcontroller doesn't support no-overlap PWM signals and your FET driver doesn't support shoot-through protection either, this drive mode might still be available to you.

Complex cycles and heat spreading

On each of the four FETs (and their catch-diodes) there are two types of loss that generate heat:

- Static loss, which is simply I^2R_{dson} times their conduction percentage per cycle for FETs, and V_fI times their conduction percentage per cycle for the catch diodes.
- Dynamic loss, which is more complex to calculate but is related to how often the device is turned on or off and how much it spends in between the 'on' and 'off' states. Dynamic loss on the catch diodes is related to the size of the shoot-through protection window and the actual turn-on and –off times of the FETs.

The average static loss between the four devices only depends on the drive mode and the duty cycle. The average dynamic loss depends on the switching frequency, the turn-on and –off times and the shoot-through protection window size. Depending on the drive mode however, the generated heat doesn't get distributed evenly across the four FETs and diodes.

We've seen that in the various drive modes, a different number of FETs are switched:

- For lock anti-phase drive, all four FETs are switching
- For sign-magnitude drive, two FETs are switching
- For asynchronous sign-magnitude drive only a single FET is switching

This means that in sign-magnitude drive, the switching FETs will have more dynamic loss (heat) than the non-switching ones. Taking static losses into consideration as well, it's easy to see that for example in our previous forward driving case Q3 and Q4 together

will dissipate more heat than Q1.

For asynchronous sign-magnitude drive, the asymmetry is even more obvious: in that case (same forward drive example) Q3 never opens, yet conducts current during the off-time through it's body-diode. As losses on such a diode are normally higher than on the FET itself, it dissipates even more heat than in the previous case.

Finally, for both types of sign-magnitude drive, there's one FET that never conducts at all.

These imbalances complicate thermal design of the bridge. It would be best if we could spread the heat more evenly across the four devices. We can do that of course mechanically, by mounting them on the same heat-sink, but there are electrical ways to make situation more balanced as well.

Lock anti-phase drive

We've just discussed that in lock anti-phase drive all four FETs are turned on and off in every cycle. The motor current is conducted through FETs for both the on- and off-time as well. This means that we can't really optimize the heat-spreading in this mode: dynamic losses are equally distributed already, and static losses depend only on the conduction time of the switches, which depends on the duty cycle (**d**). Still, let's examine the behavior of the bridge in this mode so we have something to compare the other options to.

In lock anti-phase drive the conduction times as a percentage of the cycle time are the following:

Q1	d/2
Q2	100%-d/2
Q3	100%-d/2
Q4	d/2

This is pretty much as good as it gets, with the (static) heat dissipation distributed equally among all transistors in the idle state (note that for this drive mode 50% duty cycle corresponds to idle). The further we get from the idle position the more imbalance we have between the FETs.

For dynamic losses, we'll just have to repeat the switching table and take a look at it again:

Lock anti-phase drive, FWD	**On-time**	**Off-time**

Q1_CTRL	1	0
Q2_CTRL	0	1
Q3_CTRL	0	1
Q4_CTRL	1	0

You see that – as we've said before – all four devices are turned on and off during the cycle, so each sees two transients and the associated dynamic loss.

Sign-magnitude drive

Let's take a look now at sign-magnitude drive! In our example drive above, the conduction times for the three FETs are as follows:

Q1	100%
Q2	0%
Q3	100%-d
Q4	d

In the detailed discussion we've seen that there are two possible (forward) operation modes. They differ in the way they handle the off-time current: one circulates it through the high-side transistors, and the other through the low-side ones. The low-side conducting version has the following conduction times:

Q1	d
Q2	100%-d
Q3	0%
Q4	100%

The idea is the following: let's ping-pong between the two operating modes in subsequent cycles. This way, we get the following overall conduction times:

Q1	50%+d/2
Q2	50%-d/2
Q3	50%-d/2

Q4	50%+d/2

You can see that the average conduction time is much more evenly distributed, as a matter of fact, at 0% PWM operation, all four FETs conduct for half the time on average. This averages conduction-related static loss between the FETs as much as possible, and gets to the same number that we've got for lock anti-phase drive (here the duty-cycle is between 0% and 100% for the forward direction, 0% being the idle state). The combined control looks like this:

Sign-magnitude drive, FWD, combined	**1st cycle**		**2nd cycle**	
	On-time	**Off-time**	**On-time**	**Off-time**
Q1_CTRL	1	1	1	0
Q2_CTRL	0	0	0	1
Q3_CTRL	0	1	0	0
Q4_CTRL	1	0	1	1

As you can see each FET turns on and off exactly once through two cycles, so there are two turn-on and two turn-off events in each cycle. This is the same number as the original sign-magnitude drive, which means that the overall dynamic losses are the same. However, now the dynamic losses are evenly distributed between all four FETs. Of course a similar dual-cycle drive pattern can be constructed for the reverse direction as well.

There are two down-sides to this control pattern, one fairly obvious, the other is a bit more subtle. The obvious one is that we need the ability to control the bridge differently in odd and even cycles, so we need to introduce some sort of a 'memory' or 'state' to the control circuit. If the signals are generated by HW PWM controllers inside a microcontroller, such state may or may not be possible to introduce.

The less obvious complication is that with this drive mode, there's no power-down state any more. At 0% duty cycle, the FETs are still switching, so the drive signals aren't static.

The final benefit of implementing this more combined control mode is that you can use N-channel MOSFET drivers on the high side with ease. Remember that we've discussed that these boot-strap configuration drivers can't be used to continuously open the high-side FETs. With these complex control modes, all four FETs are switching so this isn't a concern any more. The other problem with these drivers, that they can't be used with 100% duty cycle still remains, there's not much that can be done about that.

Asynchronous sign-magnitude drive

Now let's consider the other sign-magnitude drive-mode. Here we have an additional complication to consider, namely that the off-time current is conducted through one of the body diodes instead of a FET. To spread the associated heat around evenly across the four devices, a four-cycle pattern needs to be constructed:

Sign-magnitude drive, FWD, combined	**1st cycle**		**2nd cycle**		**3nd cycle**		**4nd cycle**	
	On-time	**Off-time**	**On-time**	**Off-time**	**On-time**	**Off-time**	**On-time**	**Off-time**
Q1_CTRL	1	1	1	0	1	0	1	0
Q2_CTRL	0	0	0	0	0	0	0	1
Q3_CTRL	0	0	0	1	0	0	0	0
Q4_CTRL	1	0	1	0	1	1	1	0

Here, the first and second cycle uses high-side off-time conduction, the second two cycle uses the low-side. The difference between the first and the second cycle is whether Q1 or Q3 conducts. Similarly the 3rd and 4th cycles change conduction between Q2 and Q4.

You can see that this pattern doesn't equalize dynamic loss: Q1 and Q2 switches 3/2 times per cycle on average, while Q1 and Q2 only switches 1/2 times on average. This is usually not a big problem at the switching rates of the usual H-bridge.

The bigger deal is that it distributes the heat that gets generated by the off-time current through the catch diodes evenly. That is in many cases the biggest source of heat in a bridge with asynchronous sign-magnitude drive.

Because of that, while this control pattern obviously is even more cumbersome to generate, it is even more important to do so.

Of course, just as in the previous case, we loose the inherent power-down state, so if this complex control mode is employed, the shut-down state will need to be restored by creating it externally.

Finally, just as before, this more complex cycle allows the use of boot-strapped high-side N-channel drivers, which was otherwise a limiting factor in asynchronous sign-magnitude drive.

Summary

In this article I've gone through some higher-level control problems for H-bridges. I've shown how to route the PWM signal and introduced some complex multi-cycle drive-

patterns that spread the heat around more evenly with the four FETs and their diodes that make thermal design easier. We've spent some time on discussing the need for dynamic shoot-through protection and ways to implement them as well.

The next installment of the series will focus on various safety features that a well designed H-bridge will have to employ. These involve protection for the bridge itself, for the load and the power supply, but also its mechanical environment.